A study of user level scheduling and software caching in the educational interactive system

Kaoru Tsunoda

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd-project

Recommended Citation
Tsunoda, Kaoru, "A study of user level scheduling and software caching in the educational interactive system" (1997). Theses Digitization Project. 1398.
https://scholarworks.lib.csusb.edu/etd-project/1398

This Thesis is brought to you for free and open access by the John M. Pfau Library at CSUSB ScholarWorks. It has been accepted for inclusion in Theses Digitization Project by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
A STUDY OF USER LEVEL SCHEDULING AND SOFTWARE CACHING IN THE EDUCATIONAL INTERACTIVE SYSTEM

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Kaoru Tsunoda
June 1997
A STUDY OF USER LEVEL SCHEDULING AND SOFTWARE CACHING
IN THE EDUCATIONAL INTERACTIVE SYSTEM

A Thesis
Presented to the
Faculty of
California State University,
San Bernardino

by
Kaoru Tsunoda
June 1997

Approved by:

Dr. Tung L. Yu, Chair, Computer Science
Dr. Arturo I. Concepcion
Dr. George M. Georgiou
ABSTRACT

An Educational Interactive System (EIS) is designed and implemented as a part of this study. The EIS is a text-based distance learning system which creates a virtual class on the Internet. The system has the capability of scheduling to equalize the average waiting time of the students in a class and caching to improve the system performance. Besides the implementation of the system, two major topics, scheduling and caching, are investigated in this study to discover their efficiency in the EIS.

A fixed priority multilevel queue algorithm is used to schedule students' requests. Under conditions where the requests are randomly distributed and the utilization of the server is 80%, the scheduler equalizes the average waiting time of each student in the class.

The other study shows that the high hit ratio of caching is not a critical factor for the EIS because a single cache miss operation creates an unacceptable data transmission delay as an interactive system. An ideal solution for the system is to provide a large cache in the local disk to keep the whole screen data of the session. This would reduce the network traffic.
ACKNOWLEDGEMENTS

I would like to acknowledge first to my advisor, Dr. Tong Yu, who spent a great amount time giving me advice and encouragement to accomplish this research. I would also like to acknowledge my committee members Dr. Concepcion and Dr. Georgiou for their valuable suggestions and support.

I would like to thank several graduate students who shared me a useful information: Han Sheng Yuh, Kwon Soo Han, and Jason Lin.

My special gratitude goes to my wife Hinako, for her support and great understanding.
# TABLE OF CONTENTS

ABSTRACT .................................................................................................................. iii

ACKNOWLEDGEMENTS ............................................................................................... iv

LIST OF FIGURES ....................................................................................................... vii

LIST OF TABLES ......................................................................................................... viii

CHAPTER 1. INTRODUCTION ...................................................................................... 1

1.1 Computer Conferencing .................................................................................... 1

1.2 Motivation ............................................................................................................ 2

1.3 Organization of Thesis ....................................................................................... 3

CHAPTER 2. FOUNDATION OF THE STUDY ................................................................. 5

2.1 TCP/IP Protocol ................................................................................................ 5

2.2 Scheduling ......................................................................................................... 7

2.3 Queuing Theory ................................................................................................ 8

2.4 Caching ............................................................................................................. 19

CHAPTER 3. SYSTEM DESIGN .................................................................................... 21

3.1 The Educational Interactive System ................................................................ 21

3.2 Scheduling ......................................................................................................... 25

3.3 Caching ............................................................................................................. 27

CHAPTER 4. SIMULATION ........................................................................................... 29

4.1 Scheduling ......................................................................................................... 29

4.1.1 Objective ...................................................................................................... 29

4.1.2 Simulation Methodology ............................................................................. 29

4.2 Caching ............................................................................................................. 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.1 Objective</td>
<td>33</td>
</tr>
<tr>
<td>4.2.2 Simulation Methodology</td>
<td>34</td>
</tr>
<tr>
<td>CHAPTER 5. EXPERIMENTAL AND SIMULATION RESULTS</td>
<td>37</td>
</tr>
<tr>
<td>5.1 Scheduling</td>
<td>37</td>
</tr>
<tr>
<td>5.1.1 Experimental condition</td>
<td>37</td>
</tr>
<tr>
<td>5.1.2 Results</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Caching</td>
<td>44</td>
</tr>
<tr>
<td>5.2.1 Experimental condition</td>
<td>44</td>
</tr>
<tr>
<td>5.2.2 Results</td>
<td>45</td>
</tr>
<tr>
<td>CHAPTER 6. DATA ANALYSIS</td>
<td>48</td>
</tr>
<tr>
<td>6.1 Scheduling</td>
<td>48</td>
</tr>
<tr>
<td>6.2 Caching</td>
<td>52</td>
</tr>
<tr>
<td>CHAPTER 7. DISCUSSION AND CONCLUSIONS</td>
<td>55</td>
</tr>
<tr>
<td>APPENDIX A: OUTPUT OF THE SIMULATION PROGRAMS</td>
<td>58</td>
</tr>
<tr>
<td>A.1 Scheduling</td>
<td>59</td>
</tr>
<tr>
<td>A.2 Caching</td>
<td>69</td>
</tr>
<tr>
<td>APPENDIX B: SOURCE CODE</td>
<td>74</td>
</tr>
<tr>
<td>B.1 The Educational Interactive System</td>
<td>75</td>
</tr>
<tr>
<td>B.2 Scheduling Simulation Program</td>
<td>76</td>
</tr>
<tr>
<td>B.3 Caching Simulation Program</td>
<td>85</td>
</tr>
<tr>
<td>APPENDIX C: IMPLEMENTATION OF THE EDUCATIONAL INTERACTIVE SYSTEM</td>
<td>95</td>
</tr>
<tr>
<td>ACRONYMS</td>
<td>103</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>104</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 2.1: TCP/IP network model protocol stack ............... 6
Figure 2.2: A single server multiple queuing system ......... 9
Figure 2.3: Fixed priority queues .................................. 10
Figure 3.1: Basic design of The Educational Interactive System ........................................................................ 21
Figure 3.2: Screen image of The Educational Interactive System ........................................................................ 23
Figure 4.1: Scheduling simulation with multilevel queue ................................................................................. 31
Figure 4.2: Environment of the cache simulation program ....................................................................................... 34
Figure 4.3: Algorithm of data retrieval ................................. 35
Figure 5.1: Input request dataset ........................................ 38
Figure C.1: Client/Server architecture of the Educational Interactive System ......... 96
Figure C.2: Class Diagram of the server program ............... 97
Figure C.3: Diagram of the client program routines .......... 100
LIST OF TABLES

Table 4.1: Priority condition based on “the average waiting time per talk” ........................................... 32

Table 5.1: Result of preliminary experiment, M/D/1 model ........................................................................... 41

Table 5.2: Result of M/M/1 model without priority scheduling ...................................................................... 42

Table 5.3: Result of M/M/1 model with priority scheduling ............................................................................ 43

Table 5.4: Transmission time using direct dialup to CSUSB CSCI ................................................................. 45

Table 5.5: Transmission time through the Waternet gateway .......................................................................... 46

Table 6.1: Result of 3 level priority scheduling ............... 50

Table 6.2: 3 level queue priority condition based on “the average waiting time per talk” ............................ 50

Table 6.3: Result of 7 level priority scheduling ............... 50

Table 6.4: 7 level queue priority condition based on “the average waiting time per talk” ............................ 51

Table 6.5: Average waiting time of each level .............. 52
CHAPTER 1. INTRODUCTION

1.1 COMPUTER CONFERENCING

Merging of computers and communications has been in the main stream of computer development. Interconnecting computers enhances and varies the way of computer utilization such as email system, world wide web, and video on demand.

Computer conferencing is a tool for telecommunication that reduces the need for face-to-face contact in various business and educational situations. Computer conferencing provides convenient, cost-effective interaction among people in different locations. The technology is used for such purposes as distance learning [1], virtual meetings, and collaborative work projects. A computer conferencing system connects participants to a host computer(server) through their own personal computers(clients), modems, and telephone lines or other communication links. Recent conferencing software applications allow users to send and receive not only text but also graphical images and audio data [13].
1.2 MOTIVATION

Despite the availability of some commercial computer conferencing products, there has been very little published work [7] on a systematic study of those systems. In this research, an example conferencing system, a text-based remote interactive system, an "Educational Interactive System" is designed, implemented, and examined. The system is based on the client-server architecture and TCP/IP protocol is used for the communication between the server and clients.

In the Educational Interactive System, a teacher or moderator may need to handle a lot of students' incoming requests to coordinate a class or discussion. The system also needs to achieve real-time level responses to all participants' requests in the wide area network environment.

This study focuses on two issues - scheduling and caching strategies that make the system more effective. In particular, a user level intelligent scheduler with multilevel queues is examined [9]. This supports the teacher to provide a fair opportunity for all the students in the class to participate. In addition, a software caching is
used to study the effectiveness of performance for remote access. Basics of the scheduling and caching are described in the following sections.

The goals for this research are the following:

- To research optimal scheduling algorithm for the Educational Interactive System to provide effectiveness and fairness for all the participants.
- To examine the most effective way of caching method for the system.
- To build a text-base Educational Interactive System utilizing above capabilities on the UNIX system.

1.3 ORGANIZATION OF THESIS

This paper is organized into seven chapters. Chapter 1 describes the basics of conferencing systems, the reasons of choosing these topics as well as the goals of the research. Chapter 2 describes the foundation of the study which includes the protocol used in the Educational Interactive System and queuing theory used for the mathematical approach of the scheduling. Chapter 3 explains the design of the educational interactive system, which is implemented as part
of this research. Chapter 4 discusses objectives and the
details of the simulation method for both scheduling and
caching. In Chapter 5, the results of the experimental
simulation are showed for both scheduling and caching. The
analysis of the results is made in Chapter 6. Finally, in
Chapter 7, the discussion and the conclusion and some new
related topics are presented.
CHAPTER 2. FOUNDATION OF THE STUDY

2.1 TCP/IP PROTOCOL

TCP/IP is a protocol suite that the Internet relies on. The TCP/IP protocol suite is one of many protocol suites that support the ISO/OSI communication model.[21] The well-known ISO/OSI model consists of seven layers, namely the physical layer, link layer, network layer, transport layer, session layer, presentation layer, and application layer. On the other hand, the TCP/IP protocol suite includes the Transmission Control Protocol (TCP), the Internet Protocol (IP), the User Datagram Protocol (UDP) and other protocols. Figure 2.1 shows the core relationship of protocols in the protocol suite. Although the ISO/OSI reference model defines seven layers of protocol stack, the TCP/IP network design only uses five of them.

TCP is a connection-oriented protocol that provides a reliable, full-duplex, byte stream for a user process. A byte stream type protocol treats data as a sequence of bytes regardless of the length of data. The TCP also uses a technique called virtual circuit to establish client-server communication. A virtual circuit is a point-to-point link
connection that allows computers to avoid having to choose a new route for every packet or cell. The use of a reliable TCP protocol has become the mainstream of programming of Internet applications. UDP is a connectionless protocol that has no guarantee for delivering UDP datagrams to the proper destination. A datagram type protocol treats each data unit independently. IP is the protocol located in the network layer and provides a packet delivery service for the transport layer (TCP and UDP).

![TCP/IP network model protocol stack](image)

**Figure 2.1 TCP/IP network model protocol stack**

As an Application Program Interface (API) for TCP/IP protocol based applications, the BSD socket interface was developed at UC Berkeley in the 1970s. The socket interface
includes a variety of software functions or routines to let programmers develop applications for TCP/IP networks [17].

2.2 SCHEDULING

The scheduling, usually process scheduling or CPU scheduling, is the basis of multiprogrammed operating systems [2]. By switching the CPU among processes, the operating system can increase the effectiveness of the computer. The objective of scheduling is determined by several criteria such as CPU utilization, throughput, turnaround time, waiting time, and response time [2].

There are many scheduling algorithms to determine which of the processes in the ready queue are to be assigned to the CPU. First Come, First Served Scheduling (FCFS) is the method whereby the process that requests the CPU first, gets the service of the CPU first. In Shortest Job First Scheduling (SJF), the process that has the next smallest CPU burst, gets the service next. Round Robin Scheduling (RR) is the scheme that adds the preemption to the FCFS; RR switches CPU among processes allocating to each a certain quantum (time slice). Multilevel Queue Scheduling provides several level of ready queues and the CPU is used first by
the processes in the queue with highest priority. The processes are permanently assigned to one queue. Multilevel Feedback Queue Scheduling is the same as Multilevel Queue Scheduling except that it allows processes to move between queues. Preemptive scheduling allows processes to switch from running state to ready state during the execution. On the other hand, Non-preemptive scheduling does not provide a ready state. The process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

2.3 Queuing Theory

One of the goals of this study is to justify the algorithm of a scheduling simulation program by comparing simulation results and theoretical data based on queuing theory. Queuing theory is a useful methodology for quantitative analysis of computer networks [10]. It is often used to analyze waiting time, number of events in the system, and necessary queue length [20]. A/B/m is a convenient notation for summarizing a queuing model, where A is the interarrival-time probability density, B is the service-time probability density, and m is the number of servers.
A popularly used model is the M/M/1 model (M = exponential probability density), where an exponential interarrival probability is assumed. It is a reasonable model for any system that has a large number of independent inputs such as airline reservations, file lookups on inquiries, and packet-switching networks [8]. Figure 2.2 describes the queuing system structure for a single-server with n level queues. Assume that items from queue level $k$ arrive randomly at rate $\lambda_k$ (items per second).

![Figure 2.2. A single server multiple queuing system](image)

The above multilevel queue can be considered as a fixed priority queuing. If we assume that 1 is the highest priority and n is the lowest, the queuing system can be
structured as Figure 2.3. And if the request arrivals and service-times are exponentially distributed, this model can be categorized as M/M/1 model. Thus, overall request arrival rate $\lambda$ and average waiting time $T$ can be calculated using equations just like a single server queuing model as follows.

\[
\lambda = \sum_{k=1}^{n} \lambda_k
\]

$\lambda$: mean arrival rate items per second

\[
\rho = \sum_{k=1}^{n} \rho_k = \lambda \bar{S}
\]

$\rho$: utilization

$\bar{S}$: mean service time for each arrival

\[
N = \rho / (1 - \rho)
\]

$N$: mean number of items in the system

\[
T = N / \lambda
\]

$T$: mean time an item spends in the system

![Figure 2.3 Fixed priority queues](image)

**Figure 2.3 Fixed priority queues**
[The Poisson Distribution]

Queuing theory often uses the assumption that the events causing input to the system occur at random. For example, customers who walk into a bank or users who call up an Internet provider can occur randomly at any time during the day and such events are regarded as Poisson-distributed [19]. Poisson distribution is equivalent to saying that the arrivals occur randomly or the interarrival times have an exponential distribution. It can be shown mathematically that the probability of having \( n \) arrivals in a given time period \( t \) is [10]:

\[
P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad \text{(2.3-1)}
\]

\( \lambda \): is the mean arrival rate

[Queuing theory examples]

For example, a cashier is busy 85 percent of her time and the remainder of the time she stands idle waiting for the next customer. Her utilization can be considered as 0.85. As another example, if the arm of a disk makes 9000 file references in the peak hour and the arm is in use for an average of 300 milliseconds per reference, then the utilization of the arm for the peak hour is \( \frac{9000 \times 300}{3600 \times 1000} = 0.75 \).
Finding the utilization, queuing theory will sometimes be able to give an average waiting time in the queue and the number of items in the queue and so on.

[Singe-server queuing formulas]

M/M/1 model is a simple queuing system which consists of a single server with Poisson arrivals and exponential service times. Under this condition, the utilization of the server is described as follows:

$$\rho = \frac{\lambda}{\mu} = \frac{\lambda S}{\mu} \quad (2.3-2)$$

where $\lambda$ is arrival rate, $\mu$ is service rate and $S$ is service time. The relation among $T_w$ (the time an item waits before being served), $T_s$ (the time it is being served), and $T_q$ (the time it spends in the system for both waiting and being served) is

$$T_q = T_w + T_s$$

Also $T_q$ and $T_w$ are described as follows.

$$T_q = \frac{\bar{S}}{1-\rho} \quad (2.3-3)$$

$$T_w = \frac{\rho \bar{S}}{1-\rho} \quad (2.3-4)$$
M/G/1 model is based on arbitrary or general independent service times. This means that the service time is not necessarily exponentially distributed. In this case, \( T_q \) and \( T_w \) are described as follows.

\[
T_q = \bar{S} + \frac{\rho \bar{S}A}{1 - \rho} \quad (2.3-5)
\]

\[
T_w = \frac{\rho \bar{S}A}{1 - \rho} \quad (2.3-6)
\]

where \( A = \frac{1}{2} \left[ 1 + \left( \frac{\sigma_s}{\bar{S}} \right)^2 \right] \)

These equations indicate that M/M/1 model is a special case of M/G/1 model. When the standard deviation of the service time is equal to the average, the service time distribution is considered as exponential [8,18].

There is another model called M/D/1 where the service time is constant. In this condition, \( T_q \) and \( T_w \) are:

\[
T_q = \frac{\bar{S}(2 - \rho)}{2(1 - \rho)} \quad (2.3-7)
\]

\[
T_w = \frac{\rho \bar{S}}{2(1 - \rho)} \quad (2.3-8)
\]
[Nonpreemptive priorities]

The following discussion on the derivation of the waiting time for the multilevel priority queue is taken from *Modeling and Analysis of Computer Communications Networks* by Jeremiah F. Hayes [22].

With a nonpreemptive priority queue, there is an interaction between all priority levels. Assuming a message, which has the highest priority, finds a lower priority message being served on its arrival in the system. In this situation, even if no messages in the highest priority class are in the system, there is a delay until the lower class message has completed service. It is necessary to consider no less than three priority classes to take care of the middle class being affected by both higher and lower classes. Under such a condition, assume that messages from all three classes have Poisson arrivals rate with average $\lambda_k$, $k = 1,2,3$, respectively. Let $n_k$ be the number of messages in class $k$ in the system at $i$th departure epoch.

Suppose that the $(i+1)$st departure epoch is priority class 1. In other words, a class 1 message has been assigned to the server and new messages of all three class have arrived while this message was being served. This situation can be described as follows.
\[ n_{i+1,1} = n_{i1} - 1 + a_{11} \quad (2.3-9a) \]
\[ n_{i+1,2} = n_{i2} + a_{21} \quad (2.3-9b) \]
\[ n_{i+1,3} = n_{i3} + a_{31} \quad (2.3-9c) \]

where \( n_{i1} > 0 \), \( a_{jk}, j, k = 1, 2, 3 \) is the number of messages in class \( j \) to arrive during the service of a message in class \( k \).

If the \((i+1)\)st departure is class 2,
\[ n_{i+1,1} = a_{12} \quad (2.3-10a) \]
\[ n_{i+1,2} = n_{i2} - 1 + a_{22} \quad (2.3-10b) \]
\[ n_{i+1,3} = n_{i3} + a_{32} \quad (2.3-10c) \]

where \( n_{i2} > 0 \). Because of the priority discipline, there is no message in class 1 at the \( i \)th departure.

If the \((i+1)\)st departure is class 3,
\[ n_{i+1,1} = a_{13} \quad (2.3-11a) \]
\[ n_{i+1,2} = a_{23} \quad (2.3-11b) \]
\[ n_{i+1,3} = n_{i3} - 1 + a_{33} \quad (2.3-11c) \]

where \( n_{i3} > 0 \). Since there is no message in class 1 and 2 at the \( i \)th departure.

The final equation is obtained by the situation when the \( i \)th departure leaves the system completely empty.
\[ n_{i+1,l} = a_{lk} \quad (2.3-12) \]

where \( k, l = 1, 2, 3 \) for \( n_{i1} = n_{i2} = n_{i3} = 0 \).
The probability of the above four cases are:

$$\Pi_0 = 1 - \lambda_1 \bar{S}_1 - \lambda_2 \bar{S}_2 - \lambda_3 \bar{S}_3 \quad (2.3-13a)$$
when \(n_{i1} = n_{i2} = n_{i3} = 0\).

$$\Pi_1 = \rho \frac{\lambda_1}{\lambda} \quad (2.3-13b)$$
when \(n_{i1} > 0\).

$$\Pi_2 = \rho \frac{\lambda_2}{\lambda} \quad (2.3-13c)$$
when \(n_{i1} = 0, n_{i2} > 0\).

$$\Pi_3 = \rho \frac{\lambda_3}{\lambda} \quad (2.3-13d)$$
when \(n_{i1} = n_{i2} = 0, n_{i3} > 0\).

where \(\rho = \lambda_1 \bar{S}_1 + \lambda_2 \bar{S}_2 + \lambda_3 \bar{S}_3\).

Using the conditions (2.3-9a) through (2.3-13d), calculations based on the two-dimensional probability-generating functions of \(n_{i+1,1}\) and \(n_{i+1,2}\) will result as follows.

$$\overline{n_{i1}} = 1 + \frac{\lambda_1 \sum_{k=1}^{3} \lambda_k \overline{S}_k^2}{2\rho(1 - \lambda_1 \bar{S}_1)} \quad (2.3-14)$$

$$\overline{n_{i2}} = 1 + \frac{\lambda_2 \sum_{k=1}^{3} \lambda_k \overline{S}_k^2}{2\rho(1 - \lambda_1 \bar{S}_1 - \lambda_2 \bar{S}_2)(1 - \lambda_1 \bar{S}_1)} \quad (2.3-15)$$

Where \(\overline{S}_k^2\) is the mean square service time of level \(k\). Both equations represent the expected number of messages where
one message is beginning to be served. The average number of messages which have arrived during the queuing time of the message to be served are $n_1 - 1$ for class 1 and $n_2 - 1$ for class 2. Then the average waiting time for class 1 ($T_{w1}$) and class 2 ($T_{w2}$) are derived as follows.

$$T_{w1} = \rho(n_1 - 1) = \frac{\sum_{k=1}^{3} \lambda_k \bar{S}_k^2}{2(1 - \lambda_1 \bar{S}_1)} \quad (2.3-16)$$

$$T_{w2} = \rho(n_2 - 1) = \frac{\sum_{k=1}^{3} \lambda_k \bar{S}_k^2}{2(1 - \lambda_1 \bar{S}_1)(1 - \lambda_1 \bar{S}_1 - \lambda_2 \bar{S}_2)} \quad (2.3-17)$$

Where $\rho$ is the probability of message arrivals to a nonempty system. From (2.3-16) and (2.3-17), the theoretical average waiting time of particular level for the $n$ level queue under M/G/1 condition can be calculated. The average waiting time $T_w$ of a level $j$ is:

$$T_w = \frac{\sum_{k=1}^{n} \lambda_k \bar{S}_k^2}{2(1 - \sum_{k=1}^{j} \lambda_k \bar{S}_k)(1 - \sum_{k=1}^{j} \lambda_k \bar{S}_k)} \quad (2.3-18)$$

$\lambda_k$: Request arrival rate of level $k$

$\bar{S}_k$: Average service time of level $k$

$\bar{S}_k^2$: The mean square service time of level $k$
If the service time is Poisson distribution, then

\[ S_k(t) = \mu e^{-\mu t} \]

where each level of service rate \( \mu_k = \mu \), the mean square service time of level \( k \) becomes

\[ \overline{S}_k^2 = \int_0^\infty t^2 S_k(t) dt = \int_0^\infty t^2 \mu e^{-\mu t} dt = \frac{2}{\mu^2} \]

also

\[ \overline{S}_k = \int_0^\infty t S_k(t) dt = \int_0^\infty t \mu e^{-\mu t} dt = \frac{1}{\mu} \]

from above, the mean square service time of level \( k \) becomes

\[ \overline{S}_k^2 = 2\overline{S}_k \]

Assign this to (2.3-18), then

\[ T_{wj} = \frac{\sum_{k=1}^n \lambda_k \overline{S}_k^2}{(1-\sum_{k=1}^{j-1}\lambda_k \overline{S}_k)(1-\sum_{k=1}^j\lambda_k \overline{S}_k)} \quad (2.3-19) \]

This formula is for the average waiting time under M/M/1 condition.

Since \( \overline{S}_k = \frac{1}{\mu} \), it can be also transformed into the following.

\[ T_{wj} = \frac{\sum_{k=1}^n \lambda_k \left( \frac{1}{\mu} \right)^2}{\left(1-\sum_{k=1}^{j-1}\lambda_k/\mu\right)\left(1-\sum_{k=1}^j\lambda_k/\mu\right)} \]
A cache in general is a fast storage located between the CPU and the main memory. Data are copied into the cache on a temporary basis to improve access time. When a particular piece of data is needed, it first checks whether it is in the cache. If it is, the data is used directly from the cache. If it is not, it uses the data from the main memory [11].

Cache management is a significant factor in improving the system performance; because cache size and a replacement policy may result in more than 80 percent of all accesses originally from the cache [2]. There are various replacement algorithms for software level caching. For example, FIFO algorithm simply replaces the oldest data segment. Least Recently Used (LRU) algorithm replaces the data segment that has not been used for the longest period of time. And Least Frequently Used (LFU) replacement algorithm replaces the data segment that is used least frequently.
Main memory can be considered as a cache between the CPU and the disk. This concept of caching can be applied to the network environment. If the required data segment is not in the client’s main memory as a cache, a copy of the data is brought from the server to the client system. Therefore, caching in the network environment not only decreases disk I/O, but also reduces network traffic. Moreover, if the client is located far from the server via Internet, caching becomes a more significant factor for the system performance. The study of the caching has been done in various network environments, such as distributed file systems [3,4] and world wide web servers [5,6,12,16]. A similar technique, the slave server, is used in [12] and [16] to improve response time and security for the Web server. Both approaches utilize the caching to shorten response time.
CHAPTER 3. SYSTEM DESIGN

3.1 THE EDUCATIONAL INTERACTIVE SYSTEM

A basic design of the Educational Interactive System for this research is shown in Figure 3.1.

Figure 3.1 Basic design of the Educational Interactive System.

This system is based on a centralized organization which simulates classes at school. The system consists of one server and multiple clients. As a class, one client acts as a teacher (coordinator) and the other clients perform as students. They are interconnected using TCP/IP locally.
(within intranet) or via Internet. The participants, the teacher and the students, are able to participate in the class using their PCs from their home. Ideally each client is to have extra disk space to keep every screen image of the session of the class as well as to have enough main memory to furnish an effective cache.

As shown in Figure 3.2, all participants have the same type of screen. A curses-based window is used to divide the screen into three sections. The top screen, which is the public screen, is to display the current status of the class or the previous status of the class. The middle screen, which is the private screen, is used to input individual questions, answers, or comments by the user. User inputs are sent to the server, and then distributed to all clients to be displayed on the public screen of each machine. The bottom screen, which is the guide screen, shows user commands of the system. These commands are used by the user to start, request an access to the server, and end their session in the system.
The basic procedure of the execution of the Educational Interactive System is as follows:

1. Execute the server program and specify the port number on the server machine to communicate with the clients.
2. Execute the client program on each participant's machine and specify the name of the server and the port number to establish the connection.
3. Type 'I' at the Private screen on a client's machine to initiate the session.
4. Type 'R' at the private screen on a client's machine to request sending messages to all the clients. If the server responds with the message "Start talk", the messages will be sent to all the clients and displayed on their public screens.
5. Type 'Q' at the private screen to indicate quitting the talk session.

6. Type 'E' at the private screen to terminate the session.

7. Type 'P#' (# = 1,2,3,..) to retrieve previous screen pages.

The server acts as a coordinator in the system. Upon receiving requests from the participants, the system automatically schedules them according to their priorities based on historical data. Screen data are stored temporarily in the cache of the clients as well as in the disk of the server.

All participants are able to choose to see either the current or previous screen on their public screen. When a user requests the previous screen, the image is retrieved from the cache or the disk of the server.

The UDP socket interface is used to transfer datagram between the server and clients in the system. The UDP requires easier implementation technique than the TCP socket interface does. Since the UDP does not need to make virtual connection between the server and clients, the server can handle multiple requests from many clients in a simple way. Although the UDP protocol is not reliable [21], it provides...
enough transmission capacity for the system based on low level of complexity.

The system is developed and tested under IRIX 5.3 operating system on SGI machines in the computer lab at CSUSB. The server program is written in C++ to utilize advantages such as code reuse and encapsulation. The client program is written in C, because of its simplicity. In terms of the execution of the program, the server program is executed on the server machine to provide the communication port first. Then the client programs are executed on each client machine. Upon the execution of the client program, the name of the server and the port number should be specified. The server and the client program can reside in the same machine. The typical situation is that the teacher runs both the server program and the client program on her machine and students execute the client program on their machines.

3.2 SCHEDULING

The server of the Educational Interactive System has a scheduling capability to handle students' requests. This scheduler is designed to help the teacher give a fair opportunity of participating for the students.
The system design of the scheduler depends on the definition of the criteria of the fairness and scheduling scheme. In order to implement the scheduler, the criteria of fairness must be defined. For example, if the definition of the fairness is the number of opportunities to talk, a student who had more opportunities to talk than another student gets lower priority for the next request and who had less opportunities to talk gets higher priority for the next request regardless of the total time amount of talk. If the definition of the fairness is the average waiting time per opportunity to talk, a student who has a long average waiting time per opportunity to talk gets higher priority to reduce next request’s waiting time and who has a short average waiting time per opportunity to talk gets lower priority then she tends to wait long time for the next request.

After defining the fairness for the scheduler, the type of the scheduling scheme must be chosen. Some major scheduling schemes are first-come first-served scheduling, round-robin scheduling, multilevel queue scheduling, and multilevel feedback queue scheduling.

For the scheduler of the Educational Interactive System, "the average waiting time per talk" is used for the criterion of the fairness as described in the next chapter.
And a fixed priority multilevel queue scheduling is used for the scheduling scheme. Since students in a class usually talk without interruption, the scheduling is performed in a non-preemptive way.

3.3 CACHING

As discussed in the section 2.4, the caching in a network environment is a useful technique to improve the performance of the data retrieval. Without caching, when a participant wants to see the previous screen of the class and go back to see the current screen again, the screen images would have to be retrieved from the server's disk. If the size of the screen image is large and the bandwidth of the network is limited, it may become an unacceptable duration for an interactive system. Probably, ten seconds is the maximum acceptable duration for each data retrieval for the participants [13]. When the size of data increases, the caching becomes more important for the system performance. As shown in Figure 3.1, typical cache locations in the system are the local memory system, which is a virtual memory (RAM + swap space), of the client system and server system.
The hit ratio of caching (the possibility of finding a requested data in the cache) is also a critical factor for the system with the cache. If the hit ratio is low, it does not improve or could degrade overall system performance by the overhead of the data replacement.

The Educational Interactive System is a text-based system and the size of the public screen is designed to be 960 bytes (12 x 80). However, the typical data size of screen for the web browser is 20k - 25k bytes [14] and a complex graphic based screen image may become over 1MB in size. The size of data, which is transmitted over the network, the bandwidth of the network, and the cache are interrelated to each other. Therefore, it is important to ensure the following points before applying the cache for this system.

- Is cache useful for this system?
- If so, what minimum hit ratio is required?
- Where should the cache be located?
CHAPTER 4. SIMULATION

4.1 SCHEDULING

4.1.1 Objective

An investigation of the scheduler based on the students' historical record is one of the main objectives in this study. The goal of the scheduler is to provide a fair opportunity for all the students in the class to participate.

A simulation program is implemented to determine the suitability of the scheduling algorithm for the Educational Interactive System.

4.1.2 Simulation Methodology

In order to identify the appropriate scheduling scheme for the system, the definition of fairness must be defined first. Examples of criteria are such as,

"The average waiting time per talk":

Students who have a longer waiting time per talk, than the average waiting time per talk for all students, get higher priority and those who have a shorter waiting
period per talk get lower priority. The purpose of this scheme is to equalize the average waiting time per talk for each student.

"The number of times of talking":
Students who talk many times, get lower priority and those who tend to use less opportunities to talk, get higher priority. The purpose of this scheme is to equalize the number of opportunities to talk taken by individual student.

"The total talk time":
Students who have a long total amount of talk time, get lower priority and those who have a short total amount of talk time, get higher priority. The purpose of this scheme is to equalize the total amount of talk time for each student.

In this simulation, "The average waiting time per talk" was chosen to be the criterion of the fairness. Because this criterion allows us to analyze the consistency between the experimental simulation result and theoretical result based on queuing theory. In order to equalize the average waiting time per talk for each student, a multilevel queue scheduling is used. Although the system needs to set a time limit for each student’s talk (e.g. five minutes), the
individual talk must be completed in a non-preemptive manner. Because dividing students' talk into short time quanta is not a natural way of talk in the class. As shown in Figure 4.1, a five-level queue is used for the priority scheduling simulation.

![Multilevel Queue Scheduling Diagram](image)

**Figure 4.1 Scheduling simulation with multilevel queue.**

When a student's request has arrived, the system calculates her priority based on the previous accumulated waiting time. Then the system puts the request into one of the queues with assigned priority. The requests with the highest priority are served first in a FCFS sense. If the queue is empty, the requests in the queue with the next highest priority will be served and so on. Each time, the
A request of a student in the multilevel queue is assigned to the server, the waiting time is recorded and added to the total waiting time. The number of talk and the total amount of talk time are also recorded and added to the total when the student's talk is finished. These recorded data are used to calculate the priority of the same student's next request. The execution of the program terminates within a given time limit set by the program. As a result of the execution, the program outputs the statistics of all the students including the number of opportunities to talk, the average waiting time per talk, and the total amount of talk time.

The scheduler decides priorities of the request based on the following table.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>N ≥ 1.5M</td>
<td>1</td>
</tr>
<tr>
<td>1.5M &gt; N ≥ 1.25M</td>
<td>2</td>
</tr>
<tr>
<td>1.25M &gt; N ≥ 0.75M</td>
<td>3</td>
</tr>
<tr>
<td>0.75M &gt; N ≥ 0.5M</td>
<td>4</td>
</tr>
<tr>
<td>0.5M ≥ N</td>
<td>5</td>
</tr>
</tbody>
</table>

N: The average waiting time per talk of this student.
M: The average waiting time per talk of all the students.

Table 4.1: Priority condition based on "the average waiting time per talk"
Based on this scheduling algorithm, students who have more than or equal to 150% of all the students' average waiting time get the highest priority. Students who have less than 150% and more than or equal to 125% of all the students' average waiting time get the next highest priority and so on.

4.2 CACHING

4.2.1 Objective

The main objective of the simulation is to study how the local cache and the remote cache affect the overall data transmission performance.

This simulation program is written to measure the data transmission time between the server and the client via the Internet.

The Educational Interactive System needs to transfer data among the server and the clients. When the clients request the image to appear on their screen, the image must be sent from the server within an acceptable time period. If the response time from the server is too long for the participants, they will not be able to participate in the class as an interactive mode.
4.2.2 Simulation Methodology

The program consists of a server program and a client program. As shown in Figure 4.2, the server program and the client program are executed from their individual location through a subnet or the Internet.

![Diagram of Environment of the cache simulation program.](image)

Figure 4.2. Environment of the cache simulation program.

The programs transfer pages of screen images to each other using the UDP socket interface. Both the server and the client programs create the local cache in the memory system (RAM + disk) on their execution. The screen pages
are originally kept in the server’s local disk. When screen pages are retrieved from the server, the pages are copied to the server’s cache (remote cache) and the client’s cache (local cache). During the execution of the program, if the client finds the pages in its cache, those pages are used to improve the system performance. The program based on the following algorithm is used to retrieve the pages of screen, and the Least Recently Used (LRU) algorithm is used for the page replacement in the cache.

The client requests a page of screen from the server.

If (The client finds the page in local cache)
{
    Get the page from local cache/* Local cache hit */
}
Else if (The server find the page in server’s cache)
{
    Get the page from server’s cache /* Remote cache hit */
    and also copy it to the local cache
}
Else
{
    Get the page from server’s disk /* Cache miss */
    and also copy it server’s cache and local cache
}

Figure 4.3 Algorithm of data retrieval.

The caches can hold ten pages of the screen data. The
system also keeps track of a time stamp and page number to perform LRU data replacement. The sample execution of the simulation program is described in Appendix A.2.
CHAPTER 5. EXPERIMENTAL AND SIMULATION RESULTS

5.1 SCHEDULING

5.1.1 Experimental condition

[system configuration]

The scheduling simulation program listed in Appendix B.2 can be executed on a stand alone UNIX system.

[Input dataset]

In order to create an input dataset for the experiment, the observation of classes has been conducted. This observation of the classes, csci125 and csci123 in the Computer Science Department, showed some primary features of students' talk in the classes. Those features are:

- Some students tended to talk more often than the other students did.
- The range of the talk length was from 10 seconds to around 5 minutes and the average was about 80 seconds.
- The standard deviation of the talk lengths was close to 80. When the standard deviation is equal to the mean, the distribution of the talk length is random [8].
Considering the features above, the input dataset is created as follows. The input dataset consists of three items: student identification (ID), talk length, and arrival time as shown Figure 5.1.

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>0</th>
<th>40</th>
<th>110</th>
<th>150</th>
<th>250</th>
<th>320</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
<td>5</td>
<td>12</td>
<td>7</td>
<td>17</td>
<td>0</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Talk Length</td>
<td>40</td>
<td>90</td>
<td>120</td>
<td>70</td>
<td>100</td>
<td>80</td>
<td>60</td>
</tr>
</tbody>
</table>

**Figure 5.1. Input request dataset**

*Student IDs* are in range between 0 and 29, 30 students in the class. Some students' IDs appear more often than the others in the dataset. The *talk length* is an amount of time of talk. The range of the talk length is from 10 to 270 seconds and the mean is 80 seconds. The value of the talk length is randomly selected from that range to be the standard deviation close to 80. *Arrival times* are created by a Poisson distribution using the following equation which is the probability of exactly *n* events arriving in an interval
of length $t$.

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \quad (\lambda \text{ is the mean arrival rate})$$

Using this equation, the mean arrival rate is $\lambda=0.01$ arrival per second and an interval of length is $t=1$ second. Under this condition, the probability that just one event happens within one second ($P_1(1)$) is less than 0.99%. The probability that two events happen within one second ($P_2(1)$) is less than 0.005% and so on. Then a random generator is executed every second for the whole class length to create a request arrivals dataset.

Class length used for the experiment is 100,000 seconds. The reason to choose such long class length is that the experimentation based on random events tends to require certain amount of time period or large number of input to get stable result to meet theoretical data. This is shown in the preliminary experiment in the next section. It can be considered as a class length of a whole quarter. A class is usually 90 to 120 minutes and 20 lessons in one quarter. The total amount of class length is easily beyond 100,000 seconds.
Three types of experiments were conducted. The first experiment (A) was a preliminary experiment to examine the consistency between the results of simulation program and of queuing theory. The condition of the experiment was categorized in a M/D/1 model where all talk lengths (service time) were constant. This is the simplest case of queuing model and enables us to check the validity of the simulation program. The second experiment (B) was M/M/1 model without priority scheduling. The dataset of the service time in this case was random as the input dataset described above. And the third experiment (C) was M/M/1 model with priority scheduling. A priority scheduling was added to the second experiment to observe the improvement.

5.1.2 Results

(A) Preliminary experiment, M/D/1 model.

In this experiment, the service time (talk length, $S$) was 40 seconds constant. The request arrival rate ($\lambda$) was 0.02 request/second and the utilization of the system ($\rho=\lambda S$) was 0.8. The program was executed five times for each class length to get reliable average waiting time for the requests.
<table>
<thead>
<tr>
<th>Class Length (seconds)</th>
<th>Trial</th>
<th>Number of requests</th>
<th>Average waiting time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>22</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>19</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>18.8</strong></td>
<td><strong>36.6</strong></td>
</tr>
<tr>
<td><strong>Theoretical Average</strong></td>
<td></td>
<td><strong>20.0</strong></td>
<td><strong>80.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Length (seconds)</th>
<th>Trial</th>
<th>Number of requests</th>
<th>Average waiting time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>1</td>
<td>211</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>185</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>180</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>200</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>201</td>
<td>69</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>195.4</strong></td>
<td><strong>68.6</strong></td>
</tr>
<tr>
<td><strong>Theoretical Average</strong></td>
<td></td>
<td><strong>200.0</strong></td>
<td><strong>80.0</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Length (seconds)</th>
<th>Trial</th>
<th>Number of requests</th>
<th>Average waiting time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100000</td>
<td>1</td>
<td>1989</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2028</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1968</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1993</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2045</td>
<td>77</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>2004.6</strong></td>
<td><strong>78.4</strong></td>
</tr>
<tr>
<td><strong>Theoretical Average</strong></td>
<td></td>
<td><strong>2000.0</strong></td>
<td><strong>80.0</strong></td>
</tr>
</tbody>
</table>

**Table 5.1 Result of preliminary experiment, M/D/1 model**

The theoretical average waiting time in the table is calculated using the equation (2.3-8) shown in Chapter 2. The result showed that if the class length was short like 1000 seconds, there was a significant discrepancy in the average waiting time between the theoretical result (80 seconds) and experimental result (36.6 seconds). However, as
the class length increased, the discrepancy became smaller. The experimental result of the class length 100,000 seconds reached 98.0% of the theoretical result.

**M/M/1 model without priority.**

In this experiment, the service time was 80 seconds average and randomly distributed. The request arrival rate was 0.01 request/second. The utilization of the system is 0.8. The class length was 100,000 seconds. A single level queue was used to keep waiting requests and no priority was added to the requests. The program was executed five times to get stable result as described in Appendix A. The following table shows the summary of the result.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average waiting time per request (seconds)</th>
<th>Standard deviation of average waiting time for each student</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>342</td>
<td>48.25</td>
</tr>
<tr>
<td>2</td>
<td>323</td>
<td>53.38</td>
</tr>
<tr>
<td>3</td>
<td>326</td>
<td>46.35</td>
</tr>
<tr>
<td>4</td>
<td>298</td>
<td>46.8</td>
</tr>
<tr>
<td>5</td>
<td>351</td>
<td>44.83</td>
</tr>
<tr>
<td>Average</td>
<td>328.0</td>
<td>47.92</td>
</tr>
<tr>
<td>Theoretical average</td>
<td>320.0</td>
<td>---</td>
</tr>
</tbody>
</table>

Table 5.2: Result of M/M/1 model without priority scheduling.
The result showed that the average waiting time per request was very close to its theoretical result (102.5%). The theoretical result, 320 seconds, can be calculated from the equation (2.3-4) in Chapter 2. The standard deviation of the average waiting time for each student was 47.92.

### <C> M/M/1 model with priority scheduling

The condition of the experiment was same as <B> except the addition of priority scheduling with five level queue. The results is described in Appendix A.1 and the summary is as follows.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average waiting time (seconds)</th>
<th>Standard deviation of average waiting time for each student</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>316</td>
<td>37.54</td>
</tr>
<tr>
<td>2</td>
<td>309</td>
<td>44.66</td>
</tr>
<tr>
<td>3</td>
<td>345</td>
<td>27.03</td>
</tr>
<tr>
<td>4</td>
<td>316</td>
<td>35.21</td>
</tr>
<tr>
<td>5</td>
<td>353</td>
<td>36.97</td>
</tr>
<tr>
<td>Average</td>
<td>327.8</td>
<td>36.28</td>
</tr>
</tbody>
</table>

| Theoretical average | 320.0                     | ---                                                      |

*Table 5.3: Result of M/M/1 model with priority scheduling.*

The result showed that the average waiting time per request was also very close to its theoretical result (102.4%). The standard deviation of average waiting time for
each student became 36.28 which was significantly smaller than the one without scheduling. As shown in Figure 2.3, the theoretical average waiting time for all requests can be calculated using the same method of case <B>.

5.2 CACHING

5.2.1 Experimental condition

[System Configuration]

The configurations of software and hardware of this experiment were:

Server: Hardware - SGI indigo with NFS disk
Software - IRIX 5.3(UNIX) operating system

Client: Hardware - 486DX2/66MHz, 16MB, 14.4Kb modem
Software - Linux 1.2.1

The server and client were connected via the Internet with PPP protocol.

[Transmission data size]

Four different data sizes, 1k, 2.5k, 5k, and 7.5k bytes were used. A message with size larger than 7.5k bytes could not be sent in this experiment because data transmission duration caused synchronization problem between the server
and client program. The buffer size for the transmission was 64 bytes.

[Transmission route]

Two routes were used with PPP connection as shown in Figure 4.2. Case A used an Internet service provider (WaterNet) and case B used a direct dialup to the gateway at CSUSB CSCI.

[Cache location]

The location of the remote cache was the memory system (virtual memory) of the server. The local cache was allocated in local memory system of the clients.

5.2.2 Results

The results of the transmission time for the data retrieval from the server to the client for both case A and B are described below. All measured data are the average of five times execution of the program to be more reliable result.

Case A

<table>
<thead>
<tr>
<th>Data Size (bytes)</th>
<th>1k</th>
<th>2.5k</th>
<th>5k</th>
<th>7.5k</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) Local cache hit</td>
<td>110</td>
<td>118</td>
<td>119</td>
<td>125</td>
</tr>
<tr>
<td>(B) Remote cache hit</td>
<td>821858</td>
<td>1815131</td>
<td>3459577</td>
<td>5123560</td>
</tr>
<tr>
<td>(C) Cache miss</td>
<td>825738</td>
<td>1862251</td>
<td>3534444</td>
<td>5164550</td>
</tr>
</tbody>
</table>

Table 5.4: Transmission time using direct dialup to CSUSB CSCI. (microseconds)
Case B

<table>
<thead>
<tr>
<th>Data Size (bytes)</th>
<th>1k</th>
<th>2.5k</th>
<th>5k</th>
<th>7.5k</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A) Local cache hit</td>
<td>112</td>
<td>111</td>
<td>123</td>
<td>118</td>
</tr>
<tr>
<td>(B) Remote cache hit</td>
<td>921494</td>
<td>1903730</td>
<td>3620061</td>
<td>5236214</td>
</tr>
<tr>
<td>(C) Cache miss</td>
<td>938262</td>
<td>1949748</td>
<td>3629184</td>
<td>5243740</td>
</tr>
</tbody>
</table>

**Table 5.5: Transmission time through the WaterNet gateway. (microseconds)**

(A) Local cache hit is the situation that the client found the requested data in the local cache. (B) Remote cache hit is the situation that the client found the requested data in the remote (server's) cache. (C) Cache miss is the situation that the client could not find the data in both local and remote, then needed to get it from the server's disk.

**Note:** During the execution of the experiment, no virtual memory (part of disk) usage was observed at the client as shown in the following log.

```
client:$ vmstat
procs     memory      swap      io   system   cpu
 r b w swpd free buff si so  bi  bo in   cs us  sy  id
1 0 0   0 2956   4312   0  0  17  2  183  83  5  9  87

swpd: the amount of virtual memory used (kB).
si  : Amount of memory swapped in from disk (kB/s).
so  : Amount of memory swapped to disk (kB/s).
```
From table 5.4 and 5.5, the following things were found.

- There was a little transmission delay (approximately 0.1 second) for the WaterNet gateway compared to the direct dial up. (5.2-1)

- The results of "(A)Local cache hit" were almost the same for four different data sizes for both case A and B. (5.2-2)

- The results of "(B)Remote cache hit" and "(C)Cache miss" were almost linear against the data size for both case A and B. (5.2-3)

- From (A) and (B), local cache hit creates enormous performance advantage compared to remote cache hit. (5.2-4)

- From (B) and (C), the performance difference between remote cache hit and cache miss was small; remote cache hit was only about 1% faster. (5.2-5)
CHAPTER 6. DATA ANALYSIS

6.1 SCHEDULING

The result of the preliminary experiment <A> in section 5.1.2 shows that to approach theoretical result, a certain class length is required, because randomly distributed requests get either very high density or very low density from time to time. High density request arrival creates a long waiting time and low density request arrival creates a short waiting time during that period. The experiment with the condition described in section 5.1 required 100,000 seconds for the class length to obtain a stable average waiting time. If the experimental class is too short, the average waiting time tends not to reflect the usual case.

Using a long enough class length, 100,000 seconds, the experimental average waiting time very closely approached the theoretical result (about 102.5%) for both main experimental simulations with five level queue: <B> M/M/1 model without scheduling and <C> M/M/1 model with scheduling. This proves the validity of the simulation program.
The purpose of the priority scheduling based on the criterion, "The average waiting time per talk", is to equalize the average waiting time per talk for each student. If the standard deviation (STDDEV) of the average waiting time for each student is decreased by the scheduling, the algorithm is effective. Since the STDDEV of the experiment \(<C> M/M/1 model with scheduling, 36.28, \) is less than the experiment \(<B> M/M/1 model without scheduling, 47.92, \) the priority scheduling algorithm showed an improvement.

Three and seven level queue scheduling were also examined to compare the results. The results and condition are described in Figure 6.1 through 6.4 below. Three level queue scheduling did not show an improvement (STDDEV=48.11) compared to single level queue scheduling, without scheduling. Seven level queue scheduling showed an improvement (STDDEV=44.68) but not as much as five level scheduling. This result indicates that increasing the number of queue level does not always create an improvement because it may create excessively long waiting time for the lowest priority requests.

Therefore, the priority scheduling with five-level queue based on the condition in Table 4.1 is an appropriate scheme for the fair scheduler handling the average waiting time.
<table>
<thead>
<tr>
<th>Trial</th>
<th>Average waiting time (seconds)</th>
<th>Standard deviation of average waiting time for each student</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>351</td>
<td>45.50</td>
</tr>
<tr>
<td>2</td>
<td>286</td>
<td>41.98</td>
</tr>
<tr>
<td>3</td>
<td>322</td>
<td>53.08</td>
</tr>
<tr>
<td>4</td>
<td>344</td>
<td>63.20</td>
</tr>
<tr>
<td>5</td>
<td>282</td>
<td>36.77</td>
</tr>
<tr>
<td>Average</td>
<td>317.0</td>
<td>48.11</td>
</tr>
<tr>
<td>Theoretical average</td>
<td>320.0</td>
<td>---</td>
</tr>
</tbody>
</table>

Table 6.1: Result of 3 level priority scheduling.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N \geq 1.5M )</td>
<td>1</td>
</tr>
<tr>
<td>( 1.5M &gt; N \geq 0.75M )</td>
<td>2</td>
</tr>
<tr>
<td>( 0.75M &gt; N )</td>
<td>3</td>
</tr>
</tbody>
</table>

N: The average waiting time per talk of this student.
M: The average waiting time per talk of all the students.

Table 6.2: 3 level queue priority condition based on "the average waiting time per talk"

<table>
<thead>
<tr>
<th>Trial</th>
<th>Average waiting time (seconds)</th>
<th>Standard deviation of average waiting time for each student</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>308</td>
<td>29.07</td>
</tr>
<tr>
<td>2</td>
<td>326</td>
<td>64.69</td>
</tr>
<tr>
<td>3</td>
<td>318</td>
<td>39.91</td>
</tr>
<tr>
<td>4</td>
<td>290</td>
<td>36.23</td>
</tr>
<tr>
<td>5</td>
<td>327</td>
<td>53.48</td>
</tr>
<tr>
<td>Average</td>
<td>313.8</td>
<td>44.68</td>
</tr>
<tr>
<td>Theoretical average</td>
<td>320.0</td>
<td>---</td>
</tr>
</tbody>
</table>

Table 6.3: Result of 7 level priority scheduling.
<table>
<thead>
<tr>
<th>Condition</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N \geq 2.0M )</td>
<td>1</td>
</tr>
<tr>
<td>( 2.0M &gt; N \geq 1.5M )</td>
<td>2</td>
</tr>
<tr>
<td>( 1.5M &gt; N \geq 1.25M )</td>
<td>3</td>
</tr>
<tr>
<td>( 1.25M &gt; N \geq 0.9M )</td>
<td>4</td>
</tr>
<tr>
<td>( 0.9M &gt; N \geq 0.75M )</td>
<td>5</td>
</tr>
<tr>
<td>( 0.75M &gt; N \geq 0.5M )</td>
<td>6</td>
</tr>
<tr>
<td>( 0.5M &gt; N )</td>
<td>7</td>
</tr>
</tbody>
</table>

N: The average waiting time per talk of this student.
M: The average waiting time per talk of all the students.

Table 6.4: 7 level queue condition based on "the average waiting time per talk"

[Theoretical and experimental results]

With the equation (2.3-19), the theoretical average waiting time of each level of multilevel queuing can be calculated. Let us look at the first result of priority scheduling in Appendix A, on page 64. The number of talk at the first level is 52, second level 53, third level 799, fourth level 58, and for fifth level 27. Since class length is 100,000 seconds, request arrival rates for each class are \( \lambda_1 = 0.00052 \), \( \lambda_2 = 0.00053 \), \( \lambda_3 = 0.00799 \), \( \lambda_4 = 0.00058 \), and \( \lambda_5 = 0.00027 \) respectively. The average service time of the first level is \( S_1 = 4915/52 = 94.52 \) seconds. Other levels of service time are \( S_2 = 79.53 \) seconds, \( S_3 = 79.34 \) seconds, \( S_4 = 89.31 \) seconds, and \( S_5 = 69.44 \) seconds. From this information, the average waiting time of each level becomes \( T_{w1} = 67.54 \),
The result (5.2-5) in Section 5.2.2 shows that the remote cache is not useful for this system. The result (5.2-5) also indicates that the local data copy between the memory and disk is much faster than the remote data copy over the network. If each client has the local cache in its disk to keep all the data of the session, data retrieval from the server will be eliminated.
The duration of "(A)Local cache hit" of both case A and B in section 5.2.2, is almost same for different data size. Because the cache access time is trivial compared to the duration of message display. The local cache definitely creates significant performance improvement in this kind of WAN environment. However, if the memory usage of the client is excessively heavy, unlike the condition of this experiment, it may reduce the performance improvement due to thrashing.

There is a linear relation between the size of data and the remote access time even if the size of the data is small as (5.2-3) indicates. Using the cache miss operation of case B, because of the linear relation between the data transmission time and the data size, the following equations are derived to calculate approximate data transmission time for larger data size.

\[
0.94 \text{ sec} = 1k \times A + B \quad (1)
\]
\[
5.24 \text{ sec} = 7.5k \times A + B \quad (2)
\]

from (1) and (2), \( A = 0.66, \ B = 0.28 \text{ sec.} \)

\[
Y = 0.66X + 0.28 \quad (6.2-1)
\]

where \( y \) is duration(sec), \( x \) is data size.

If an acceptable data transmission time is 10 seconds \((Y = 10)\), the maximum data size will be about 15k bytes \((x = 14.7)\). This indicates that one page of text-base screen
(about 1k bytes) can be transferred fast enough to be an interactive mode without any cache.

Assuming that the client's local cache hit ratio is 80% and one page of screen data is 25k bytes, users will find 80% of time of screen image retrieval without any problem because of (5.2-4). However, 20% of time they need to wait more than 15 seconds and this is not tolerable as an interactive system. This indicates that high hit ratio of cache is not a critical factor for the Educational Interactive System because a single cache miss operation could cause unacceptable data transmission delay.

If the size of data is 20k to 25k bytes like web pages, larger bandwidth is required to transmit data as an interactive system. It is also better to provide a large enough cache in the local disk to keep all the screen data from the server. An additional experiment was conducted to test the data transmission from the client’s local disk to its memory. It showed that 1 MB of data can be transferred from the local disk to the local memory (no page fault were found during the experiment) in around 0.5 second.
CHAPTER 7. DISCUSSION AND CONCLUSIONS

Two main objectives were investigated in this study: the efficiency and optimization of the scheduling and caching for the Educational Interactive System.

For the scheduling part of this study, we specifically used a fixed priority five level queue algorithm. The purpose of the scheduling is to equalize the average waiting time of each student in the class. When the utilization of the server is 0.8 and class length is 100,000 seconds, the average waiting time of each student in the class showed an improvement by using the priority scheduling. The standard deviation of the waiting time of each student decreased from 47.92 to 36.28. This indicates that the five level queue algorithm is efficient under this condition. With three level and seven level queue priority scheduling, improvement of the scheduling was not as much as the one with five level queue. Therefore, among single, three, five, and seven level queue, the five level queue scheduling was optimal in this experiment.

The other topic, the experimental simulation of caching, showed interesting results. We found that the location of caching is a more important factor than the
replacement algorithm because the Educational Interactive System requires a real-time system-level response to the users. If the response from the server is unacceptably slow, users no longer participate in the class properly. We assumed that ten seconds is the maximum tolerable duration for the screen image transmission of the system. Under such a condition, a remote cache hardly made any performance improvement for the system (1% improvement compared to without the remote cache). Although the local cache created significant improvement for cache hit operation, a single cache miss operation created a critical time delay for the data transmission. As a result, all the screen images sent from the server should be kept in the local disk of all the clients. 1 MB of image can be transmitted to the screen buffer of the client within 1.0 second with this configuration. It could also replace the allocation of both local and remote cache in the memory.

The experiment showed that although caches improve system performance, a text-based Educational Interactive System is not necessary to have caches to achieve interactive capability. However, as the screen image increases like a web page, the bandwidth of the network needs to be larger than this experimental condition. Ideally
the Educational Interactive System should utilize the cache in the local disk.

Lastly, it is necessary to note that this experiment was conducted with the current level of hardware configuration. As time goes by, CPU power, network bandwidth, and Internet technologies will be enhanced at a fast pace. Then the result of this experiment may be very different from the one today.
APPENDIX A: OUTPUT OF THE SIMULATION PROGRAMS
A.1 SCHEDULING

<B> M/M/1 Model Without Priority Scheduling

/* Scheduling Simulation Log */
/* */
/* - Average talk length 80 seconds */
/* - Request arrival density 0.01 request/second */
/* - Class length 100,000 seconds */

/***** [ Without Priority Scheduling Trial 1 ] **********

*****[ Summary ]*********************************************************************

NumTalk[ 0]: 32 ServiceT: 2480 WaitingT: 9330 AveWaitingT: 291 ServiceT Ave: 77
NumTalk[ 1]: 25 ServiceT: 2200 WaitingT: 11361 AveWaitingT: 454 ServiceT Ave: 88
NumTalk[ 3]: 39 ServiceT: 2545 WaitingT: 14354 AveWaitingT: 368 ServiceT Ave: 65
NumTalk[ 4]: 33 ServiceT: 2925 WaitingT: 11275 AveWaitingT: 341 ServiceT Ave: 88
NumTalk[ 5]: 28 ServiceT: 2490 WaitingT: 9452 AveWaitingT: 337 ServiceT Ave: 88
NumTalk[ 6]: 28 ServiceT: 2560 WaitingT: 8880 AveWaitingT: 317 ServiceT Ave: 91
NumTalk[ 7]: 37 ServiceT: 3695 WaitingT: 12056 AveWaitingT: 325 ServiceT Ave: 99
NumTalk[ 8]: 43 ServiceT: 3785 WaitingT: 16143 AveWaitingT: 375 ServiceT Ave: 88
NumTalk[ 9]: 37 ServiceT: 3270 WaitingT: 15193 AveWaitingT: 410 ServiceT Ave: 88
NumTalk[10]: 27 ServiceT: 2115 WaitingT: 10923 AveWaitingT: 404 ServiceT Ave: 78
NumTalk[13]: 37 ServiceT: 3400 WaitingT: 12749 AveWaitingT: 344 ServiceT Ave: 91
NumTalk[14]: 34 ServiceT: 3575 WaitingT: 10475 AveWaitingT: 308 ServiceT Ave: 105
NumTalk[15]: 41 ServiceT: 3150 WaitingT: 14570 AveWaitingT: 355 ServiceT Ave: 76
NumTalk[16]: 30 ServiceT: 2365 WaitingT: 10076 AveWaitingT: 335 ServiceT Ave: 78
NumTalk[17]: 25 ServiceT: 1915 WaitingT: 6867 AveWaitingT: 274 ServiceT Ave: 76
NumTalk[18]: 35 ServiceT: 3000 WaitingT: 9196 AveWaitingT: 262 ServiceT Ave: 85
NumTalk[19]: 38 ServiceT: 2895 WaitingT: 14778 AveWaitingT: 388 ServiceT Ave: 76
NumTalk[20]: 37 ServiceT: 2665 WaitingT: 12858 AveWaitingT: 347 ServiceT Ave: 72
NumTalk[21]: 34 ServiceT: 2670 WaitingT: 12149 AveWaitingT: 357 ServiceT Ave: 78
NumTalk[22]: 35 ServiceT: 2685 WaitingT: 11114 AveWaitingT: 317 ServiceT Ave: 76
NumTalk[23]: 28 ServiceT: 2505 WaitingT: 10978 AveWaitingT: 392 ServiceT Ave: 89
NumTalk[24]: 35 ServiceT: 2735 WaitingT: 11072 AveWaitingT: 306 ServiceT Ave: 76
NumTalk[25]: 33 ServiceT: 2425 WaitingT: 10411 AveWaitingT: 315 ServiceT Ave: 73
NumTalk[27]: 33 ServiceT: 2525 WaitingT: 11037 AveWaitingT: 314 ServiceT Ave: 76
NumTalk[28]: 29 ServiceT: 2865 WaitingT: 11525 AveWaitingT: 337 ServiceT Ave: 98
NumTalk[29]: 37 ServiceT: 3165 WaitingT: 12591 AveWaitingT: 340 ServiceT Ave: 85

< Total > NumTalk : 993
AveNumTalk : 33
ServiceTime : 84920
WaitingTime : 339645
AveWaitingTime: 342
ServiceTimeAve: 2830

< Priority Level Information >

Level: 1 # of Talk: 993 Talk Time: 84920 Waiting Time: 339645
Level: 2 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 3 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 4 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 5 # of Talk: 0 Talk Time: 0 Waiting Time: 0

59
/* Scheduling Simulation Log */
/* - Average talk length 80 seconds */
/* - Request arrival density 0.01 request/second */
/* - Class length 100,000 seconds */

/* Without Priority Scheduling Trial 2 */

<table>
<thead>
<tr>
<th>NumTalk</th>
<th>ServiceT</th>
<th>WaitingT</th>
<th>AveWaitingT</th>
<th>ServiceT Ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31</td>
<td>3135</td>
<td>10705</td>
<td>345</td>
</tr>
<tr>
<td>1</td>
<td>36</td>
<td>2545</td>
<td>10755</td>
<td>298</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>2800</td>
<td>7299</td>
<td>197</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>2250</td>
<td>8825</td>
<td>353</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>2850</td>
<td>11368</td>
<td>392</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>2125</td>
<td>11515</td>
<td>329</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>2280</td>
<td>8356</td>
<td>278</td>
</tr>
<tr>
<td>7</td>
<td>30</td>
<td>2085</td>
<td>10797</td>
<td>359</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>2070</td>
<td>9479</td>
<td>338</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>2880</td>
<td>13319</td>
<td>332</td>
</tr>
<tr>
<td>10</td>
<td>32</td>
<td>3225</td>
<td>12382</td>
<td>386</td>
</tr>
<tr>
<td>11</td>
<td>40</td>
<td>3335</td>
<td>8498</td>
<td>212</td>
</tr>
<tr>
<td>12</td>
<td>39</td>
<td>3220</td>
<td>11677</td>
<td>299</td>
</tr>
<tr>
<td>13</td>
<td>32</td>
<td>2635</td>
<td>10906</td>
<td>340</td>
</tr>
<tr>
<td>14</td>
<td>33</td>
<td>1950</td>
<td>11716</td>
<td>355</td>
</tr>
<tr>
<td>15</td>
<td>38</td>
<td>2700</td>
<td>9352</td>
<td>246</td>
</tr>
<tr>
<td>16</td>
<td>39</td>
<td>2735</td>
<td>12006</td>
<td>307</td>
</tr>
<tr>
<td>17</td>
<td>29</td>
<td>1945</td>
<td>10268</td>
<td>354</td>
</tr>
<tr>
<td>18</td>
<td>25</td>
<td>1645</td>
<td>10108</td>
<td>404</td>
</tr>
<tr>
<td>19</td>
<td>36</td>
<td>2735</td>
<td>15216</td>
<td>422</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>2070</td>
<td>9155</td>
<td>366</td>
</tr>
<tr>
<td>21</td>
<td>30</td>
<td>2170</td>
<td>10410</td>
<td>347</td>
</tr>
<tr>
<td>22</td>
<td>39</td>
<td>3850</td>
<td>12398</td>
<td>317</td>
</tr>
<tr>
<td>23</td>
<td>32</td>
<td>2815</td>
<td>11801</td>
<td>368</td>
</tr>
<tr>
<td>24</td>
<td>41</td>
<td>3250</td>
<td>10701</td>
<td>261</td>
</tr>
<tr>
<td>25</td>
<td>36</td>
<td>2895</td>
<td>10990</td>
<td>305</td>
</tr>
<tr>
<td>26</td>
<td>41</td>
<td>3330</td>
<td>13191</td>
<td>321</td>
</tr>
<tr>
<td>27</td>
<td>37</td>
<td>2830</td>
<td>14459</td>
<td>390</td>
</tr>
<tr>
<td>28</td>
<td>32</td>
<td>2320</td>
<td>9028</td>
<td>282</td>
</tr>
<tr>
<td>29</td>
<td>39</td>
<td>2910</td>
<td>11979</td>
<td>307</td>
</tr>
</tbody>
</table>

< Total >
NumTalk: 1016
AveNumTalk: 33
ServiceTime: 79585
WaitingTime: 328659
AveWaitingTime: 323
ServiceTimeAve: 2652

< Priority Level Information >

Level: 1 # of Talk: 1016 Talk Time: 79585 Waiting Time: 328659
Level: 2 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 3 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 4 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 5 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Scheduling Simulation Log

- Average talk length 80 seconds
- Request arrival density 0.01 request/second
- Class length 100,000 seconds

Without Priority Scheduling Trial 3

<table>
<thead>
<tr>
<th>NumTalk</th>
<th>ServiceTime</th>
<th>WaitingTime</th>
<th>AveWaitingTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4430</td>
<td>12637</td>
<td>274</td>
</tr>
<tr>
<td>1</td>
<td>2175</td>
<td>12250</td>
<td>382</td>
</tr>
<tr>
<td>2</td>
<td>2105</td>
<td>8701</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>1550</td>
<td>9730</td>
<td>347</td>
</tr>
<tr>
<td>4</td>
<td>2965</td>
<td>11320</td>
<td>290</td>
</tr>
<tr>
<td>5</td>
<td>2635</td>
<td>7840</td>
<td>261</td>
</tr>
<tr>
<td>6</td>
<td>1920</td>
<td>10219</td>
<td>364</td>
</tr>
<tr>
<td>7</td>
<td>3605</td>
<td>11164</td>
<td>328</td>
</tr>
<tr>
<td>8</td>
<td>1930</td>
<td>8420</td>
<td>300</td>
</tr>
<tr>
<td>9</td>
<td>2360</td>
<td>7275</td>
<td>346</td>
</tr>
<tr>
<td>10</td>
<td>3245</td>
<td>10791</td>
<td>308</td>
</tr>
<tr>
<td>11</td>
<td>2255</td>
<td>10691</td>
<td>334</td>
</tr>
<tr>
<td>12</td>
<td>2295</td>
<td>10023</td>
<td>303</td>
</tr>
<tr>
<td>13</td>
<td>2590</td>
<td>9852</td>
<td>317</td>
</tr>
<tr>
<td>14</td>
<td>2225</td>
<td>8819</td>
<td>275</td>
</tr>
<tr>
<td>15</td>
<td>3745</td>
<td>9309</td>
<td>265</td>
</tr>
<tr>
<td>16</td>
<td>2430</td>
<td>9762</td>
<td>336</td>
</tr>
<tr>
<td>17</td>
<td>2710</td>
<td>13981</td>
<td>349</td>
</tr>
<tr>
<td>18</td>
<td>2525</td>
<td>11223</td>
<td>362</td>
</tr>
<tr>
<td>19</td>
<td>2120</td>
<td>10653</td>
<td>409</td>
</tr>
<tr>
<td>20</td>
<td>2190</td>
<td>10739</td>
<td>325</td>
</tr>
<tr>
<td>21</td>
<td>2670</td>
<td>12745</td>
<td>439</td>
</tr>
<tr>
<td>22</td>
<td>2605</td>
<td>6658</td>
<td>277</td>
</tr>
<tr>
<td>23</td>
<td>2310</td>
<td>10265</td>
<td>353</td>
</tr>
<tr>
<td>24</td>
<td>2825</td>
<td>9173</td>
<td>295</td>
</tr>
<tr>
<td>25</td>
<td>2010</td>
<td>11342</td>
<td>420</td>
</tr>
<tr>
<td>26</td>
<td>3610</td>
<td>11174</td>
<td>302</td>
</tr>
<tr>
<td>27</td>
<td>3555</td>
<td>11484</td>
<td>310</td>
</tr>
<tr>
<td>28</td>
<td>3150</td>
<td>13286</td>
<td>316</td>
</tr>
<tr>
<td>29</td>
<td>1900</td>
<td>10307</td>
<td>396</td>
</tr>
</tbody>
</table>

Total NumTalk: 954
AveNumTalk: 31
ServiceTime: 78640
WaitingTime: 311833
AveWaitingTime: 326
ServiceTimeAve: 2621

Priority Level Information:

- Level 1: 954 Talk Time: 78640 Waiting Time: 311833
- Level 2: 0 Talk Time: 0 Waiting Time: 0
- Level 3: 0 Talk Time: 0 Waiting Time: 0
- Level 4: 0 Talk Time: 0 Waiting Time: 0
- Level 5: 0 Talk Time: 0 Waiting Time: 0

61
--- [Without Priority Scheduling Trial 4] ---

**Summary**

```plaintext
NumTalk[ 0] : 24 ServiceT:1765 WaitingT: 7618 AveWaitingT: 317 ServiceTave: 73
```

< Total > NumTalk : 981
   AveNumTalk : 32
   ServiceTime : 78640
   WaitingTime : 293091
   AveWaitingTime: 298
   ServiceTimeAve: 2621

< Priority Level Information >

<table>
<thead>
<tr>
<th>Level</th>
<th># of Talk</th>
<th>Talk Time</th>
<th>Waiting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>981</td>
<td>78640</td>
<td>293091</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Scheduling Simulation Log

- Average talk length 80 seconds
- Request arrival density 0.01 request/second
- Class length 100,000 seconds

----------[ Without Priority Scheduling Trial 5]----------

********[ Summary ]********

< Total > NumTalk: 1023
   AveNumTalk: 34
   ServiceTime: 81030
   WaitingTime: 359094
   AveWaitingTime: 351
   ServiceTimeAve: 2701

< Priority Level Information >

Level: 1 # of Talk: 1023 Talk Time: 81030 Waiting Time: 359094
Level: 2 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 3 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 4 # of Talk: 0 Talk Time: 0 Waiting Time: 0
Level: 5 # of Talk: 0 Talk Time: 0 Waiting Time: 0
<C> M/M/1 Model With Priority Scheduling

/***************************************************************************/
/* Scheduling Simulation Log */
/***************************************************************************/
/* - Average talk length 80 seconds */
/* - Request arrival density 0.01 request/second */
/* - Class length 100,000 seconds */
/***************************************************************************/

//--[ With Priority Scheduling Trial 1 ]-------------------------------------*/

*****[ Summary ]*******************************************************/

-> Priorioty based on Average Waiting Time with Level 5

NumTalk[ 0]: 28 ServiceT:2400 WaitingT:10053 AveWaitingT: 359 ServiceTave: 85
NumTalk[ 1]: 35 ServiceT:2830 WaitingT: 9994 AveWaitingT: 285 ServiceTave: 80
NumTalk[ 2]: 34 ServiceT:2605 WaitingT:11119 AveWaitingT: 327 ServiceTave: 76
NumTalk[ 3]: 24 ServiceT:2080 WaitingT: 8893 AveWaitingT: 370 ServiceTave: 86
NumTalk[ 4]: 33 ServiceT:2970 WaitingT:11689 AveWaitingT: 354 ServiceTave: 90
NumTalk[ 5]: 26 ServiceT:2025 WaitingT: 8702 AveWaitingT: 334 ServiceTave: 77
NumTalk[ 6]: 30 ServiceT:2345 WaitingT:10324 AveWaitingT: 344 ServiceTave: 78
NumTalk[ 7]: 32 ServiceT:3060 WaitingT: 9756 AveWaitingT: 304 ServiceTave: 95
NumTalk[ 8]: 31 ServiceT:2725 WaitingT:10466 AveWaitingT: 337 ServiceTave: 87
NumTalk[ 9]: 32 ServiceT:2760 WaitingT:12103 AveWaitingT: 378 ServiceTave: 86
NumTalk[10]: 34 ServiceT:2750 WaitingT:13118 AveWaitingT: 385 ServiceTave: 80
NumTalk[13]: 38 ServiceT:3535 WaitingT:10861 AveWaitingT: 285 ServiceTave: 93
NumTalk[14]: 32 ServiceT:2425 WaitingT:11202 AveWaitingT: 350 ServiceTave: 75
NumTalk[15]: 25 ServiceT:2260 WaitingT: 9671 AveWaitingT: 386 ServiceTave: 90
NumTalk[16]: 41 ServiceT:4115 WaitingT:12270 AveWaitingT: 299 ServiceTave: 100
NumTalk[17]: 32 ServiceT:2445 WaitingT: 9660 AveWaitingT: 301 ServiceTave: 76
NumTalk[18]: 28 ServiceT:1305 WaitingT: 9998 AveWaitingT: 357 ServiceTave: 46
NumTalk[19]: 29 ServiceT:1795 WaitingT: 7771 AveWaitingT: 267 ServiceTave: 61
NumTalk[20]: 36 ServiceT:2130 WaitingT:10171 AveWaitingT: 282 ServiceTave: 59
NumTalk[21]: 42 ServiceT:3645 WaitingT:11197 AveWaitingT: 266 ServiceTave: 86
NumTalk[22]: 34 ServiceT:2780 WaitingT: 9388 AveWaitingT: 276 ServiceTave: 81
NumTalk[23]: 33 ServiceT:1840 WaitingT:10035 AveWaitingT: 304 ServiceTave: 55
NumTalk[24]: 31 ServiceT:2210 WaitingT:10303 AveWaitingT: 332 ServiceTave: 71
NumTalk[25]: 36 ServiceT:2555 WaitingT: 9142 AveWaitingT: 253 ServiceTave: 70
NumTalk[26]: 38 ServiceT:2825 WaitingT:11902 AveWaitingT: 315 ServiceTave: 74
NumTalk[27]: 33 ServiceT:3115 WaitingT:10711 AveWaitingT: 324 ServiceTave: 94
NumTalk[28]: 46 ServiceT:4135 WaitingT:13402 AveWaitingT: 291 ServiceTave: 89
NumTalk[29]: 37 ServiceT:3310 WaitingT:10545 AveWaitingT: 285 ServiceTave: 89

< Total > NumTalk : 989
    AveNumTalk : 32
    ServiceTime :79575
    WaitingTime :313252
    AveWaitingTime: 316
    ServiceTimeAve: 2652

< Priority Level Information >

Level: 1 # of Talk:  52 Talk Time:  4915 Waiting Time:  3492
Level: 2 # of Talk:  53 Talk Time:  4215 Waiting Time:  3815
Level: 3 # of Talk:  799 Talk Time:  63390 Waiting Time:  207530
Level: 4 # of Talk:  58 Talk Time:  5180 Waiting Time:  57935
Level: 5 # of Talk:  27 Talk Time:  1875 Waiting Time:  40480
/**
 * Scheduling Simulation Log
 */
/**
 * - Average talk length 80 seconds
 * - Request arrival density 0.01 request/second
 * - Class length 100,000 seconds
 */

/* [ With Priority Scheduling Trial 2] */

> Priority based on Average Waiting Time with Level 5

<table>
<thead>
<tr>
<th>Level</th>
<th># of Talk</th>
<th>Talk Time</th>
<th>Waiting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>69</td>
<td>6895</td>
<td>4060</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
<td>6130</td>
<td>5507</td>
</tr>
<tr>
<td>3</td>
<td>759</td>
<td>58960</td>
<td>201760</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
<td>5170</td>
<td>50272</td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>3650</td>
<td>47584</td>
</tr>
</tbody>
</table>

< Total >
NumTalk   : 998
AveNumTalk: 33
ServiceTime: 80805
WaitingTime: 309183
AveWaitingTime: 309
ServiceTimeAve: 2693

< Priority Level Information >
Scheduling Simulation Log

- Average talk length 80 seconds
- Request arrival density 0.01 request/second
- Class length 100,000 seconds

[With Priority Scheduling Trial 3]

--- Priority based on Average Waiting Time with Level 5

Priority Level Information

Level: 1 # of Talk: 62 Talk Time: 4350 Waiting Time: 3661
Level: 2 # of Talk: 40 Talk Time: 3365 Waiting Time: 2240
Level: 3 # of Talk: 763 Talk Time: 63695 Waiting Time: 196563
Level: 4 # of Talk: 69 Talk Time: 5465 Waiting Time: 65366
Level: 5 # of Talk: 48 Talk Time: 2925 Waiting Time: 71367
/****** [ With Priority Scheduling Trial 4 ] ----------------------------------------*/

***** [ Summary ]***************************************************************************/

-> Priority based on Average Waiting Time with Level 5

NumTalk[ 0]: 28 ServiceT:2175 WaitingT: 9486 AveWaitingT: 338 ServiceT Ave: 77
NumTalk[ 1]: 30 ServiceT:2205 WaitingT: 9220 AveWaitingT: 307 ServiceT Ave: 73
NumTalk[ 2]: 29 ServiceT:2435 WaitingT: 9648 AveWaitingT: 332 ServiceT Ave: 83
NumTalk[ 3]: 34 ServiceT:2110 WaitingT:11545 AveWaitingT: 339 ServiceT Ave: 62
NumTalk[ 4]: 36 ServiceT:2375 WaitingT:13270 AveWaitingT: 368 ServiceT Ave: 65
NumTalk[ 5]: 37 ServiceT:2810 WaitingT:12480 AveWaitingT: 337 ServiceT Ave: 75
NumTalk[ 6]: 35 ServiceT:2765 WaitingT:11745 AveWaitingT: 335 ServiceT Ave: 79
NumTalk[ 7]: 35 ServiceT:2780 WaitingT: 8899 AveWaitingT: 254 ServiceT Ave: 79
NumTalk[ 9]: 34 ServiceT:2650 WaitingT:10463 AveWaitingT: 307 ServiceT Ave: 77
NumTalk[10]: 28 ServiceT:2415 WaitingT: 9770 AveWaitingT: 348 ServiceT Ave: 86
NumTalk[12]: 31 ServiceT:2395 WaitingT: 9894 AveWaitingT: 319 ServiceT Ave: 77
NumTalk[13]: 37 ServiceT:3390 WaitingT:11747 AveWaitingT: 317 ServiceT Ave: 91
NumTalk[14]: 28 ServiceT:1615 WaitingT: 7877 AveWaitingT: 281 ServiceT Ave: 65
NumTalk[15]: 31 ServiceT:2120 WaitingT: 7833 AveWaitingT: 252 ServiceT Ave: 68
NumTalk[16]: 38 ServiceT:3335 WaitingT:13208 AveWaitingT: 347 ServiceT Ave: 87
NumTalk[17]: 33 ServiceT:2210 WaitingT:10354 AveWaitingT: 313 ServiceT Ave: 66
NumTalk[18]: 41 ServiceT:4570 WaitingT:11250 AveWaitingT: 274 ServiceT Ave: 111
NumTalk[19]: 25 ServiceT:2395 WaitingT: 8924 AveWaitingT: 356 ServiceT Ave: 95
NumTalk[20]: 36 ServiceT:2580 WaitingT:12469 AveWaitingT: 346 ServiceT Ave: 71
NumTalk[21]: 28 ServiceT:2050 WaitingT: 9632 AveWaitingT: 344 ServiceT Ave: 73
NumTalk[22]: 30 ServiceT:2275 WaitingT:10981 AveWaitingT: 366 ServiceT Ave: 75
NumTalk[23]: 31 ServiceT:2620 WaitingT:10941 AveWaitingT: 352 ServiceT Ave: 84
NumTalk[24]: 31 ServiceT:2215 WaitingT: 8416 AveWaitingT: 271 ServiceT Ave: 71
NumTalk[26]: 27 ServiceT:2755 WaitingT: 8728 AveWaitingT: 323 ServiceT Ave: 102
NumTalk[28]: 31 ServiceT:2480 WaitingT:11063 AveWaitingT: 356 ServiceT Ave: 80
NumTalk[29]: 39 ServiceT:3585 WaitingT:10302 AveWaitingT: 264 ServiceT Ave: 91

< Total > NumTalk : 976
AveNumTalk : 32
ServiceTime : 77930
WaitingTime : 309187
AveWaitingTime: 316
ServiceTime Ave: 2597

< Priority Level Information >

Level: 1 # of Talk:  85  Talk Time:  6685  Waiting Time:  5612
Level: 2 # of Talk:  67  Talk Time:  5315  Waiting Time:  6423
Level: 3 # of Talk:  671  Talk Time:  54195  Waiting Time:  148030
Level: 4 # of Talk:  77  Talk Time:  6065  Waiting Time:  68848
Level: 5 # of Talk:  76  Talk Time:  5670  Waiting Time:  80274

67
/* Scheduling Simulation Log */
/* */
/* - Average talk length 80 seconds */
/* - Request arrival density 0.01 request/second */
/* - Class length 100,000 seconds */
/* */
/Z----------[ With Priority Scheduling Trial 5 ]-------------------------------*/

*****[ Summary ]**************************************************************************

-> Priority based on Average Waiting Time with Level 5

<table>
<thead>
<tr>
<th>NumTalk</th>
<th>ServiceT</th>
<th>WaitingT</th>
<th>AveWaitingT</th>
<th>ServiceT Ave</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
<td>2330</td>
<td>16746</td>
<td>418</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>3040</td>
<td>15440</td>
<td>386</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>2730</td>
<td>13042</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>39</td>
<td>3015</td>
<td>14899</td>
<td>382</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>2285</td>
<td>15415</td>
<td>405</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>2760</td>
<td>10264</td>
<td>405</td>
</tr>
<tr>
<td>6</td>
<td>37</td>
<td>3525</td>
<td>11343</td>
<td>306</td>
</tr>
<tr>
<td>7</td>
<td>33</td>
<td>2685</td>
<td>12476</td>
<td>378</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>2910</td>
<td>13258</td>
<td>414</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>3155</td>
<td>11415</td>
<td>380</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>2760</td>
<td>15415</td>
<td>405</td>
</tr>
<tr>
<td>11</td>
<td>37</td>
<td>3015</td>
<td>14899</td>
<td>382</td>
</tr>
<tr>
<td>12</td>
<td>39</td>
<td>2285</td>
<td>15415</td>
<td>405</td>
</tr>
<tr>
<td>13</td>
<td>38</td>
<td>2330</td>
<td>16746</td>
<td>418</td>
</tr>
<tr>
<td>14</td>
<td>29</td>
<td>2135</td>
<td>9243</td>
<td>318</td>
</tr>
<tr>
<td>15</td>
<td>29</td>
<td>2770</td>
<td>9545</td>
<td>329</td>
</tr>
<tr>
<td>16</td>
<td>34</td>
<td>2690</td>
<td>12647</td>
<td>371</td>
</tr>
<tr>
<td>17</td>
<td>38</td>
<td>2825</td>
<td>12546</td>
<td>330</td>
</tr>
<tr>
<td>18</td>
<td>23</td>
<td>1745</td>
<td>7606</td>
<td>330</td>
</tr>
<tr>
<td>19</td>
<td>34</td>
<td>2195</td>
<td>10933</td>
<td>321</td>
</tr>
<tr>
<td>20</td>
<td>41</td>
<td>4225</td>
<td>14077</td>
<td>343</td>
</tr>
<tr>
<td>21</td>
<td>30</td>
<td>2495</td>
<td>10105</td>
<td>336</td>
</tr>
<tr>
<td>22</td>
<td>27</td>
<td>2060</td>
<td>10138</td>
<td>375</td>
</tr>
<tr>
<td>23</td>
<td>27</td>
<td>1930</td>
<td>10494</td>
<td>388</td>
</tr>
<tr>
<td>24</td>
<td>36</td>
<td>2805</td>
<td>10761</td>
<td>298</td>
</tr>
<tr>
<td>25</td>
<td>33</td>
<td>2110</td>
<td>11536</td>
<td>349</td>
</tr>
<tr>
<td>26</td>
<td>32</td>
<td>2240</td>
<td>11392</td>
<td>356</td>
</tr>
<tr>
<td>27</td>
<td>30</td>
<td>2185</td>
<td>10428</td>
<td>347</td>
</tr>
<tr>
<td>28</td>
<td>45</td>
<td>3540</td>
<td>17232</td>
<td>382</td>
</tr>
<tr>
<td>29</td>
<td>42</td>
<td>2540</td>
<td>12974</td>
<td>308</td>
</tr>
<tr>
<td>30</td>
<td>38</td>
<td>3060</td>
<td>14175</td>
<td>373</td>
</tr>
</tbody>
</table>

< Total > NumTalk : 1039
AveNumTalk : 34
ServiceTime : 80800
WaitingTime : 366965
AveWaitingTime: 353
ServiceTimeAve: 2693

< Priority Level Information >

Level: 1 # of Talk:  76 Talk Time:  6575 Waiting Time:  6165
Level: 2 # of Talk:  56 Talk Time:  3480 Waiting Time:  4842
Level: 3 # of Talk:  805 Talk Time:  62735 Waiting Time:  242382
Level: 4 # of Talk:  58 Talk Time:  3835 Waiting Time:  45219
Level: 5 # of Talk:  44 Talk Time:  4175 Waiting Time:  66357
A.2 CACHING

/***************************************************************
/* Caching simulation log 1 */
/* */
/* - Transmission data size 2.5k */
/* */
/* - Direct dialup to CSUSB CSCI gateway */
/* */
/***************************************************************

/*-----[ Server log ]-------------------------------------------*/

<indigo>$ server
Enter port number: 5500
msg: 4 Miss
msg: 6 Miss
Total cache hit : 0
Total cache miss: 2
Total hit ratio : 0%
Simulation is done!

<indigo>$

/*-----[ Client log] -------------------------------------------*/

<PC486>$ client
Miss
start: 857332363.363017
durat: 1799066 micro /* transmission time */
end : 857332365.162083

Hit
start: 857332365.162336
durat: 120 micro
end : 857332365.162456

Hit
start: 857332365.162646
durat: 112 micro
end : 857332365.162758

Hit
start: 857332365.162946
durat: 111 micro
end : 857332365.163057

Hit
start: 857332365.163243
durat: 113 micro
end : 857332365.163356

Hit
start: 857332365.163544
durat: 113 micro
end : 857332365.163657
Miss
start: 857332365.163843
durat: 1798265 micro

Hit
start: 857332366.962359
durat: 123 micro

Hit
start: 857332366.962670
durat: 115 micro

Hit
start: 857332366.962972
durat: 115 micro

Hit
start: 857332366.963273
durat: 114 micro

Hit
start: 857332366.963574
durat: 725 micro

Total cache hit: 10
Total cache miss: 2
Total hit ratio: 83%
Simulation is done!

<PC486>$
Enter port number: 5500

msg: 1 Miss
msg: 2 Miss
msg: 3 Miss
msg: 4 Miss
msg: 5 Miss
msg: 6 Miss
msg: 1 Hit
msg: 2 Hit
msg: 3 Hit
msg: 4 Hit
msg: 5 Hit
msg: 6 Hit
msg: 1 Hit
msg: 2 Hit
msg: 3 Hit

Total cache hit : 9
Total cache miss: 6
Total hit ratio : 60%
Simulation is done!

<indigo>$

Ubuntu$ client

Miss
start: 857332557.336333
end : 857332559.185560
durat: 1849227 micro

Miss
start: 857332559.185865
end : 857332560.985494
durat: 1799629 micro

Miss
start: 857332560.985735
end : 857332562.795352
durat: 1809617 micro

Miss
start: 857332562.795594
end : 857332564.670039
durat: 1874445 micro
Miss
start: 857332564.670280
durat: 1835196 micro
end : 857332566.505476

Miss
Cache replace at 0
start: 857332566.505717
durat: 1799775 micro
end : 857332568.305492

Miss
Cache replace at 1
start: 857332568.305730
durat: 1813339 micro
end : 857332570.119069

Miss
Cache replace at 2
start: 857332570.119309
durat: 1871470 micro
end : 857332571.990779

Miss
Cache replace at 3
start: 857332571.991023
durat: 1834888 micro
end : 857332573.825911

Miss
Cache replace at 4
start: 857332573.826155
durat: 1873867 micro
end : 857332575.700022

Miss
Cache replace at 0
start: 857332575.700899
durat: 1899350 micro
end : 857332577.600249

Miss
Cache replace at 1
start: 857332577.600489
durat: 1848766 micro
end : 857332579.449255

Miss
Cache replace at 2
start: 857332579.449496
durat: 1870824 micro
end : 857332581.320320

Miss
Cache replace at 3
start: 857332581.320564
durat: 1848750 micro
end : 857332583.169314
Miss
Cache replace at 4
start: 857332583.169554
derat: 1850393 micro
end : 857332585.19947
Total cache hit : 0
Total cache miss: 15
Total hit ratio : 0%
Simulation is done!

<PC486>$
APPENDIX B: SOURCE CODE
B.1 The Educational Interactive System

The source code is located under
/u/class/tongyu/thesis/kaoru on orion. Notes are written in
README file in the directory.
B.2 Scheduling Simulation Program

/*-----[ att.h ]---------------------------------------------------------------*/

#include "define.h"

class Attend
{
  private:
    int NumTalk[NumOfStudents];
    int ServiceTime[NumOfStudents];
    int WaitingTime[NumOfStudents];
    int Priority[NumOfStudents];
    int LevelWaitingTime[5];
    int LevelTalkTime[5];
    int LevelNumOfTalk[5];
    int AveServiceTime[NumOfStudents];
    int AveWaitingTime[NumOfStudents];
    int TotalNumTalk;
    int TotalServiceTime;
    int TotalWaitingTime;
    int AveTotalNumTalk;
    int AveTotalServiceTime;
    int AveTotalWaitingTime;

  public:
    Attend()
    {
      TotalNumTalk=0;
      TotalServiceTime=0;
      TotalWaitingTime=0;
      AveTotalNumTalk=0;
      AveTotalServiceTime=0;
      AveTotalWaitingTime=0;

      for(int i=0; i<NumOfStudents; i++)
        {
          NumTalk[i]=0;
          ServiceTime[i]=0;
          WaitingTime[i]=0;
          AveServiceTime[i]=0;
          AveWaitingTime[i]=0;
        }
    }

    int CheckPriority(int Sid) { return(Priority[Sid]); };
    int IncrementNumTalk(int Sid);
    int AddServiceTime(int Sid, int TTime);
    int AddWaitingTime(int Sid, int TTime);
    void CalcAverage(void);
    void CalcTotalAverage(void);
    int CalcPriority(int Sid, int Level);
    int SetPriority(int Sid, int Pri) { return(Priority[Sid] = Pri); };
    void AddLevelTime(int Prio, int Wt, int Tt);
    void PrtLevelTotal(void);
    void InitPriority(int Level);
```cpp
void PrtAttendee(int Type);

class Random
{
private:
    double Interval;
    double ArrivalRate;
    double P0;
    double P1;
    double P2;
    double E;
    double tmp;

    int Count;
    int random;

    fstream OutStream;

public:
    Random() {
        const char RequestFile[] = "input.dat";
        const char ErrorMsg[] = "Unable to open file: ";

        srand48( (unsigned) time (NULL) );
        srand( (unsigned) time (NULL) );

        E = 2.71828;
        Count = 0;

        Interval = 1.0;
        ArrivalRate = 0.01;
        tmp = ArrivalRate * Interval;
        P0 = pow(E, -(tmp));
        P1 = tmp * P0;
        P2 = 1.0 - P0;

        OutStream.open(RequestFile, ios::out);
        if(OutStream.fail())
        {
            cerr << ErrorMsg << RequestFile << endl<<endl;
            exit(-1);
        }
    }

    ~Random(){
        OutStream.close();
    }

    int CheckReqArrival(void);
    int GetUid(void);
    Rdata *SetReqData(int Id, int Ts);
    int NumOfEvents(void);
};
```

/****-[ queue.h ]--------------------------------------------------------------------------------****/

#include "define.h"
#include "data.h"

class Node {
  public:
    Rdata *Ptr;
    Node *Next;

    Node(Rdata *P) {
      Ptr=P;
      Next = NULL;
    }
};

class ReqQ {
  private:
    Node *Head[NumOfPriority];
    Node *Tail[NumOfPriority];
    int TotalItem;

  public:
    ReqQ() {
      for(int i; i<NumOfPriority; i++) {
        Head[i] = NULL;
        Tail[i] = NULL;
      }
      TotalItem = 0;
    }

    int Append(int Prio, Rdata *P);
    int IsEmpty(void);
    Node *Pickup(int *Pri);
    int Lookup(int Id);
};

/#-----------------[ main.cc ]---------------------------------------------------------------*/

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include "rand.h"
#include "att.h"

main( int argc, char** argv )
{
    int ClassLength;
    int PrioLevel;
    int Etype;
    int Uid;
    int Priority = 0;
    int Priority2 = 0;

    Rdata *Rptr;
    Node *Nptr;

    int TimeStamp = 0;
    int TalkTime;
    int EndTime;
int ArrivalTime;
int WaitingTime;
int CurrentSpeaker = -1;

Attend Student;
ReqQ Q;
Random Rand;

if(argc != 3 )
{
    fprintf(stderr, "Usage: %s class_length prio_level\n", *argv);
    exit(1);
}

ClassLength = atoi(argv[1]);
PrioLevel = atoi(argv[2]);

Student.InitPriority(PrioLevel);

while(1)
{
    if( (ClassLength*1.2) <= TimeStamp )
        break;

    if( ClassLength >= TimeStamp )
    {
        Etype = Rand.CheckReqArrival();
        if( Etype == 1 )
        {
            Uid = Rand.GetUid();
            if( (CurrentSpeaker != Uid) && (!Q.Lookup( Uid )) )
                break;
            } while(1);

            if(PrioLevel > 1) {
                Priority = Student.CalcPriority(Uid, PrioLevel);
                //cout << "Uid: " << Uid << " " << Priority << endl;
            }
            Rptr = Rand.SetReqData(Uid, TimeStamp);
            Q.Append( Priority, Rptr);
        }
        else if( Etype == 2 )
        {
            Uid = Rand.GetUid();
            if( (CurrentSpeaker != Uid) && (!Q.Lookup( Uid )) )
                break;
            } while(1);

            if(PrioLevel > 1) {
                Priority = Student.CalcPriority(Uid, PrioLevel);
            }
            Rptr = Rand.SetReqData(Uid, TimeStamp);
            Q.Append( Priority, Rptr);
        }
    }

    do {
        Uid = Rand.GetUid();
        if( (CurrentSpeaker != Uid) && (!Q.Lookup( Uid )) )
            break;
        } while(1);
if(PrioLevel > 1) {
    Priority = Student.CalcPriority(Uid, PrioLevel);
}

Rptr = Rand.SetReqData(Uid, TimeStamp);
Q.Append( Priority, Rptr);
}

if( EndTime == TimeStamp )
{
    CurrentSpeaker = -1;
}

if( CurrentSpeaker == -1 && !Q.IsEmpty() )
{
    Nptr = Q.Pickup( &Priority2 );
    CurrentSpeaker = Nptr->Ptr->Id;
    ArrivalTime = Nptr->Ptr->Ts;
    WaitingTime = 0;
    if( ArrivalTime < TimeStamp )
    {
        WaitingTime = TimeStamp - ArrivalTime;
        Student.AddWaitingTime(CurrentSpeaker, WaitingTime);
    }
    Student.IncrementNumTalk(CurrentSpeaker);
    TalkTime = Nptr->Ptr->Tt;
    Student.AddServiceTime(CurrentSpeaker, TalkTime);
    Student.CalcAverage();
    Student.CalcTotalAverage();
    EndTime = TimeStamp + TalkTime;
    TimeStamp++;
}

cout << "# " << Rand.NumOfEvents() << endl;
Student.PrtAttendee(PrioLevel);
Student.PrtLevelTotal();

*/-----[ att.cc ]---------------------------------------------------------------*/
TotalServiceTime = TotalServiceTime + TTime;
return(ServiceTime[Sid]);
}

int Attend::AddWaitingTime(int Sid, int TTime)
{
    TotalWaitingTime = TotalWaitingTime + TTime;
    return(WaitingTime[Sid]);
}

void Attend::PrtAttendee(int Level)
{
    cout << "
*****[ Summary ]**************************
";
    cout << "**************************
";
    if(Level == 1)
    {
        cout << endl << "    -> No Priority" << endl;
    }
    else
    {
        printf("\n    -> Priority based on Average Waiting Time with Level %d\n", Level);
    }
    for(int i=0; i<NumOfStudents; i++)
    {
        printf("NumTalk[%2d]:%3d", i, NumTalk[i]);
        printf(" ServiceT:%4d", ServiceTime[i]);
        printf(" WaitingT:%5d", WaitingTime[i]);
        printf(" AveWaitingT:%4d", AveWaitingTime[i]);
        printf(" ServiceTave:%4d", AveServiceTime[i]);
    }
    printf("\n < Total >");
    printf(" NumTalk :%5d\n", TotalNumTalk);
    printf(" AveNumTalk :%5d\n", AveTotalNumTalk);
    printf(" ServiceT :%5d\n", TotalServiceTime);
    printf(" WaitingTime :%5d\n", TotalWaitingTime);
    printf(" AveWaitingTime:%5d\n", AveTotalWaitingTime);
    printf(" ServiceTave:%5d\n", AveTotalServiceTime);
    cout << endl;
}

void Attend::CalcAverage(void)
{
    for(int i=0; i<NumOfStudents; i++)
    {
        if(NumTalk[i] != 0) {
            AveServiceTime[i] = (int) ServiceTime[i]/NumTalk[i];
            AveWaitingTime[i] = (int) WaitingTime[i]/NumTalk[i];
        }
    }
}

void Attend::CalcTotalAverage(void)
{
    if(TotalNumTalk != 0) {
        AveTotalServiceTime = (int) TotalServiceTime/NumOfStudents;
        AveTotalWaitingTime = (int) TotalWaitingTime/TotalNumTalk;
        AveTotalNumTalk = (int) TotalNumTalk/NumOfStudents;
        }
```cpp
int Attend::CalcPriority(int Sid, int Level)
{
    if (Level == 3)
    {
        if (AveTotalWaitingTime != 0 && NumTalk[Sid] != 0)
        {
            if (AveWaitingTime[Sid] >= (AveTotalWaitingTime * 1.5))
                return(0);
            else if (AveWaitingTime[Sid] < (AveTotalWaitingTime * 0.75))
                return(2);
        }
    return(1);
    }
    else if (Level == 5)
    {
        if (AveTotalWaitingTime != 0 && NumTalk[Sid] != 0)
        {
            if (AveWaitingTime[Sid] >= (AveTotalWaitingTime * 1.25))
                return(0);
            else if (AveWaitingTime[Sid] >= (AveTotalWaitingTime * 1.1))
                return(1);
            else if (AveWaitingTime[Sid] < (AveTotalWaitingTime * 0.75))
                return(4);
            else if (AveWaitingTime[Sid] < (AveTotalWaitingTime * 0.9))
                return(3);
        }
    return(2);
    }
}

void Attend::AddLevelTime(int Prio, int Wt, int Tt)
{
    LevelTalkTime[Prio] = LevelTalkTime[Prio] + Tt;
    LevelNumOfTalk[Prio]++;
}

void Attend::PrtLevelTotal(void)
{
    printf(" < Priority Level Information >\n\n");
    for (int i=0; i<5; i++)
    {
        printf("Level: %d # of Talk: %3d, %i, LevelNumOfTalk[i]);
        printf("Talk Time: %5d", LevelTalkTime[i]);
        printf("Waiting Time: %5d\n", LevelWaitingTime[i]);
    }
}

void Attend::InitPriority(int Level)
{
    if (Level==3) {
        for (int i=0; i<NumOfStudents; i++)
            Priority[i]=1;
    }
    else if (Level==5) {
        for (int i=0; i<NumOfStudents; i++)
            Priority[i]=2;
    }
}
```
```cpp
#include <iostream.h>
#include <stdlib.h>
#include "rand.h"

int Random::CheckReqArrival(void)
{
    double Tmp;
    while( (Tmp = lrand48()) > 10000001)
        continue;
    Tmp = Tmp/10000000;
    if( Tmp <= P1 ) {
        return(1);
    } else if( Tmp < P2 ) {
        return(2);
    } else
        return(0);
}

int Random::GetUid(void)
{
    int Tmp;
    while( ( random = rand() ) >= 30000 )
        continue;
    random = random/100;
    Tmp = random/10;
    return(Tmp);
}

Rdata *Random::SetReqData(int Id, int Ts)
{
    int TalkTable[10]={10,15,20,25,45,65,80,110,170,270};
    int Tmp;
    Rdata *Ptr = new Rdata();
    Tmp = random/10;
    Tmp = random - Tmp*10;
    PTR->Id = Id;
    PTR->Ts = Ts;
    PTR->Tt = TalkTable[Tmp];
    OutStream << PTR->Tt << " ";
    Count++;
    if( 0 == (Count%5) )
        OutStream << endl;
    return(PTR);
}
```
```c
#include <stdio.h>
#include "queue.h"

int Random::NumOfEvents(void)
{
    return(Count);
}

int ReqQ::Append(int Prio, Rdata *P)
{
    Node *Tmp = new Node(P);
    if(Head[Prio] == NULL) {
        Head[Prio] = Tail[Prio] = Tmp;
    } else {
        Tail[Prio]->Next = Tmp;
        Tail[Prio] = Tmp;
    }
    TotalItem++;
    if (TotalItem >= 29) {
        cout << "Q is full" << endl;
        exit(0);
    }
}

int ReqQ::IsEmpty(void)
{
    return(TotalItem == 0);
}

Node *ReqQ::Pickup(int *Pri)
{
    Node *Tmp;
    for(int i=0; i<NumOfPriority; i++)
    {
        if(Head[i] != NULL) {
            Tmp = Head[i];
            Head[i] = Head[i]->Next;
            TotalItem--;
            *Pri = i;
            return(Tmp);
        }
    }
    return(NULL);
}
```
B.3 Caching Simulation Program

/*--------[ server.h ]-------------------------------*/

#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>

#define SMALL 2
#define MEDIUM 5
#define LARGE 10
#define VLARGE 15
#define FSMALL 1024
#define FMEDIUM 2560
#define F LARGE 5120
#define FVLARGE 7680
#define FILESIZE FMEDIUM
#define MAXLINE 512
#define LINES MIDUM
#define COLUMNS MAXLINE+1
#define MAXCACHE 10
#define FALSE 0
#define TRUE 1

struct cache {
    int tag;
    char page[LINES][COLUMNS];
    int tstamp;
};

int time_stamp;
int used_pos;
int total_hit;
int total_miss;
int total_ratio;

struct cache my_cache[MAXCACHE];
char *fname="storage";

int establish(int *sfd, struct sockaddr_in *s_addr);

/*--------[ server.c ]-------------------------------*/

#include <stdio.h>
#include <sys/time.h>
#include <string.h>

#include <netdb.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>
#include "server.h"
int main()
{
    int sockfd; /* socket descriptor */
    struct sockaddr_in serv_addr; /* server's address */
    struct sockaddr_in cli_addr; /* client's address */

    int clilen; /* client's address size */
    char msg[MAXLINE]; /* buffer for message */
    int msglen; /* message length */
    int number; /* page number of data */
    int c_pos; /* cache position */

    total_hit = 0;
    total_miss = 0;
    total_ratio = 0;

    time_stamp = 0;
    used_pos = 0;

    /* establish UDP connection with client */
    establish(&sockfd, &serv_addr);

    /* recieve page number from client and return page to client */
    for(;;)
    {
        memset(&msg, 0, sizeof(msg)); /* initialize buffer */
        clilen = sizeof(cli_addr); /* set client's address length */

        /* recieve page number from client */
        msglen = recvfrom(sockfd, msg, MAXLINE, 0,
                          (struct sockaddr *)&cli_addr, &clilen);
        if(msglen<0)
            perror("recvfrom error");

        printf("msg:\%3s", msg);
        number = atoi(msg);

        /* if end sign(999), finish program */
        if(number == 999)
        {
            print_result();
            printf("Simulation is done!\n");
            exit(0);
        }

        c_pos = check_cache(number); /* check cache data is there or not */
        if(c_pos != -1) /* hit, send page from cache to client */
        {
            printf(" Hit\n");
            send_page_to_client(c_pos, &sockfd, &cli_addr, &clilen);
            total_hit++; /* increment hit count */
        } else /* miss, get page from disk to cache */
        {
            printf(" Miss\n");
            if(used_pos<MAXCACHE) /* cache is not full yet */
            {
                c_pos = used_pos; /* available cache position */

                /* get page from disk to cache */
                get_page_from_disk(number, c_pos);
            } else /* cache is full */
            {
                printf(" Cache is full\n");
                exit(1);
            }
        }
    }
}
/* send page from cache to client */
send_page_to_client(c_pos, sockfd, &cli_addr, &clilen);

used_pos++; /* increment cache used position */
total_miss++; /* for cold start */
}
else /* cache is full, replace it */
{
    printf("Cache replace\n");
    /* choose replacing cache position using LRU policy */
c_pos = least_recently_used();
    /* get page from disk to cache */
    get_page_from_disk(number, c_pos, sockfd, &serv_addr);
    /* send page from cache to client */
    send_page_to_client(c_pos, sockfd, &cli_addr, &clilen);
total_miss++; /* increment miss count */
}

int establish(int *sfd, struct sockaddr_in *s_addr)
{
    int p_number;

    printf("\nEnter port number: ");
    scanf("%d", &p_number);

    memset(s_addr, 0, sizeof(struct sockaddr_in));
    /* bzero((char *)s_addr, sizeof(struct sockaddr_in)); */
    if(*sfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        perror("server: can't open datagram socket");
    s_addr->sin_family = AF_INET;
    s_addr->sin_addr.s_addr = htonl(INADDR_ANY);
    s_addr->sin_port = htons(p_number);
    if(bind(*sfd, (struct sockaddr *)s_addr, sizeof(struct sockaddr_in)) < 0)
        perror("server: can't bind local address");
}

int check_cache(int num)
{
    int i;

    for(i=0; i<used_pos; i++) /* check out the cache */
    {
        if(my_cache[i].tag==num) /* hit, return location */
            return(i);
    }
    return(-1); /* miss */
}

int get_page_from_disk(int pnum, int epos)
{
    FILE *fp;
int i;
int spoint;
char buf[MAXLINE+1];

/* open storage file with read binary mode */
if ( (fp=fopen(fname, "rb")) == NULL )
{
    perror(fname);
    exit(1);
}

fseek(fp, 0L, 0); /* set pointer to the beginning of the file */
spoint = FILESIZE * (pnum-1); /* calculate offset of accessing page */
fseek(fp, spoint, 0); /* forward pointer to the page */

for(i=0; i<LINES; i++)
{
    fread(buf, sizeof(char), MAXLINE, fp);
    buf[MAXLINE] = '\0';
    strcpy(my_cache[cpos].page[i], buf);
    /* printf("c[%d].p[%2d]: %s", cpos, i, my_cache[cpos].page[i]); */
}
my_cache[cpos].tstamp = time_stamp++;
my_cache[cpos].tag = pnum;

close(fp);

int send_page_to_client(int cpos, int *sd, struct sockaddr *c_addr, int *c_len)
{
    char msg[MAXLINE];
    int msglen;
    int i;

    for(i=0; i<LINES; i++)
    {
        memset(&msg, 0, sizeof(msg));
        strncpy(msg, my_cache[cpos].page[i], MAXLINE);
        msglen = strlen(msg);
        if( sendto(*sd, msg, msglen, 0, c_addr, *c_len) != msglen)
            perror("sendto error");
    }
}

int print_cache(int cpos)
{
    int i;
    for(i=0; i<LINES; i++)
    {
        printf("c[%d].p[%2d]: %s", cpos, i, my_cache[cpos].page[i]);
    }
}

int least_recently_used()
{
    int i;
    int ts;
    int pos=0;

    ts = my_cache[0].tstamp;
    for(i=1; i<MAXCACHE; i++)
    {
```c
if(my_cache[i].tstamp<ts) {
    ts = my_cache[i].tstamp;
    pos = i;
}
return(pos);
}

int prt_result(void)
{
    int tmp;
    printf("\nTotal cache hit:%3d\n", total_hit);
    printf("Total cache miss:%3d\n", total_miss);
    tmp = total_hit + total_miss;
    printf("Total hit ratio:%3d%%\n", (total_hit * 100)/tmp);
}

#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#define SMALL 2 /* 512 X 2 = 1K */
#define MIDIUM 5 /* 512 X 5 = 2.5K */
#define LARGE 10 /* 512 X 10 = 5K */
#define VLARGE 15 /* 512 X 15 = 7.5K */

/* storage file size */
#define FSMALL 1024
#define FMIDIUM 2560
#define FLARGE 5120
#define FVLARGE 7680
#define FILESIZE FMIDIUM
#define MAXLINE 512 /* length of line of the cache */
#define LINES MIDIUM /* number of lines of the cache */
#define COLUMNS MAXLINE+1 /* column size for the cache */
#define MAXQUEUE 10 /* length of queue for most used */
#define MAXCACHE 5 /* cache size */
#define FALSE 0
#define TRUE 1

/* one cell of cache */
struct cache {
    int tag; /* page number of data */
    char page[LINES][COLUMNS]; /* one page of data */
    int tstamp; /* time stamp */
    int count; /* the number of hit */
};
/* queue for least used policy only */
int update_q[MAXQUEUE];
int q_head;
int full_q;

int timer; /* timer */
```
```c
int used_pos;     /* cache used level */
int total_hit;    /* total number of hit */
int total_miss;   /* total number of miss */
int total_ratio;  /* total hit ratio */

struct cache my_cache[MAXCACHE]; /* actual cache declaration */
char *fname="storage";           /* data file name */

int establish(int *sfd, struct sockaddr_in *s_addr);
int get_page_from_server(int num, int pos, int *sfd, struct sockaddr_in *s_addr);
int prt_cache(int cnum);
int check_cache(int num);
int least_recently_used();
int least_used();
int update_cache(int num);
int prt_result(void);

.getOrElse{ client.cc }

#include <stdio.h>
#include <sys/time.h>
#include <string.h>
#include <netdb.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "client.h"

void main()
{
    int number;    /* page number */
    struct timeval ts;   /* variable for gettimeofday() */
    int c_pos;      /* cache position */
    int start_sec;  /* start time in second */
    int start_usec; /* start time in micro second */
    int end_sec;    /* end time in second */
    int end_usec;   /* end time in micro second */
    int duration;   /* duration of data retrieval */
    int sockfd;     /* socket descriptor */
    struct sockaddr_in serv_addr;   /* server address */

    total_hit = 0;
    total_miss = 0;
    total_ratio = 0;

    timer = 0;  /* timer set to 0 */
    used_pos = 0;  /* set cache empty */
    q_head = 0;   /* queue head at 0 */
    full_q = FALSE;  /* set queue is not full */

    /* establish UDP connection with server */
    establish(&sockfd, &serv_addr);
    /* retrieve data until end */
    for(;;)
}
```
{/* printf("Which page you need [1-10] ? "); */
scanf("%d", &number); /* input the page number to retrieve */

/* set starting time */
gmtimefd(&ts, NULL);
start_sec = ts.tv_sec;
start_usec = ts.tv_usec;

c_pos = check_cache(number); /* check page is in cache or not */
if(c_pos != -1) /* hit, get page from cache */
{
    printf("Hit");
    prt_cache(c_pos); /* print out page in cache */
    /* update_cache(number); for least used policy */

    /* set ending time */
gmtimefd(&ts, NULL);
end_sec = ts.tv_sec;
end_usec = ts.tv_usec;

total_hit++; /* increment hit count */
}
else /* cache miss, need to get page from server */
{
    printf("Miss");
    if(used_pos<MAXCACHE) /* local cache is not full */
    {
        c_pos = used_pos; /* set available cache position */

        /* get page from server */
        get_page_from_server(number, c_pos, &sockfd, &serv_addr);
        prt_cache(c_pos); /* print page in cache */

        /* set ending time */
gmtimefd(&ts, NULL);
end_sec = ts.tv_sec;
end_usec = ts.tv_usec;
used_pos++; /* increment cache position */
total_miss++; /* cold start */
    }
else /* local cache is full, need to replace it */
    {
        /* choose replacing cache position using LRU policy */
        c_pos = least_recently_used();
        printf(" Cache replace at %d\n", c_pos);

        /* get page from server */
        get_page_from_server(number, c_pos, &sockfd, &serv_addr);
        prt_cache(c_pos);

        /* set ending time */
gmtimefd(&ts, NULL);
end_sec = ts.tv_sec;
end_usec = ts.tv_usec;

total_miss++; /* increment miss count */
}
int establish( int *sfd, struct sockaddr_in *s_addr )
{
    char hostname[50];
    int p_number;
    struct hostent *hp;

    struct sockaddr_in cli_addr;

    strcpy(hostname, "indigo");

    /* printf("Enter port number: "); */
    /* scanf("%d", &p_number); */
    p_number = 5500;

    memset( s_addr, 0, sizeof(struct sockaddr_in));
    if( (hp = gethostbyname( hostname ) ) == NULL ) {
        perror("gethostbyname error");
        return(-1);
    }

    if( ( *sfd = socket(AF_INET, SOCK_DGRAM, 0) ) < 0 ) {
        perror("socket error");
        return(-1);
    }

    memset( s_addr, 0, sizeof(struct sockaddr_in));
    memcpy( &(s_addr->sin_addr), hp->h_addr, hp->h_length );
    s_addr->sin_family = AF_INET;
    s_addr->sin_port = htons( (u_short) p_number );

    memset( (char *)&cli_addr, 0, sizeof(struct sockaddr_in));
    cli_addr.sin_family = AF_INET;
    cli_addr.sin_addr.s_addr = htonl( INADDR_ANY );
    cli_addr.sin_port = htons(0);

    if( bind( *sfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr)) < 0 ) {
        perror("bind error");
        return(-1);
    }
    return(0);
}

int get_page_from_server(int num, int cpos, int *sd, struct sockaddr_in *s_addr)
{
    char msg[MAXLINE];
    int msglen;
    int s_len;
    int i;
    char buf[MAXLINE+1];

    memset(&msg, 0, sizeof(msg));
    sprintf(msg, "%d%c", num, '\0');
msglen = strlen(msg);
if( sendto(*sd, msg, msglen, 0, (struct sockaddr *)s_addr,
    sizeof(*s_addr)) != msglen)
    perror("sendto error");
if(!strcmp(msg, "999") ) {
    prt_result();
    printf("Simulation is done!
");
    exit(0);
}
for(i=0; i<LINES; i++) {
    memset(&msg, 0, sizeof(msg));
    msglen = recvfrom(*sd, buf, MAXLINE, 0,
        (struct sockaddr*)s_addr, &s_len);
    if(msglen<0)
        perror("recvfrom error");
    buf[MAXLINE]=\0;
    strcpy(my_cache[epos].page[i], buf);
}
    my_cache[epos].tag=num;
    my_cache[epos].count=0;
}
int prt_cache(int cpos)
{
    int i;
    for(i=0; i<LINES; i++) {
        /* printf("c[%d].p[%2d]: %s", epos, i, my_cache[cpos].page[i]); */
    }
    printf(" %d lines at %d printed!
", LINES, epos);
    my_cache[cpos].tstamp = timer++;
    my_cache[cpos].count++;
}
int check_cache(int num)
{
    int i;
    for(i=0; i<used_pos; i++) /* check out the cache */
    {
        if(my_cache[i].tag==num)
            return(i);
    }
    return(-1);
}
int least_recently_used()
{
    int i;
    int ts;
    int pos=0;
    ts = my_cache[0].tstamp;
    for(i=1; i<MAXCACHE; i++) {
        if(my_cache[i].tstamp<ts) {
            ts = my_cache[i].tstamp;
            pos = i;
        }
    }
    return(pos);
}
int least_used()
{
    int cnt;
    int i;
    int pos=0;
    cnt = my_cache[0].count;
    for(i=1; i<MAXCACHE; i++) {
        if(my_cache[i].count<cnt) {
            cnt = my_cache[i].count;
            pos=i;
        }
    }
    return(pos);
}

int update_cache(int num)
{
    int i;
    if(full_q != TRUE) {
        update_q[q_head] = num;
        q_head++;
        if(q_head == MAXQUEUE) {
            q_head = 0;
            full_q = TRUE;
        }
    } else {
        my_cache[update_q[q_head]].count--;
        update_q[q_head] = num;
        q_head++;
        if(q_head == MAXQUEUE) {
            q_head = 0;
        }
    }
    for(i=0; i<q_head; i++)
        printf("%d ",update_q[i]);
    printf("\n");
}

int prt_result(void)
{
    int tmp;
    printf("Total cache hit:%3d\n", total_hit);
    printf("Total cache miss:%3d\n", total_miss);
    tmp = total_hit + total_miss;
    printf("Total hit ratio: %3d%%\n", (total_hit * 100)/tmp );
}
APPENDIX C: IMPLEMENTATION OF THE EDUCATIONAL INTERACTIVE SYSTEM
The Educational Interactive System consists of a server and client program. The basic architecture is described in Figure C.1. A single server handles multiple requests from clients simultaneously. When the server receives a message from the client, it responds as the message requested. Both the server and client program are event-driven execution and communicate each other by UDP socket interface.

![Diagram of Client/Server architecture](image)

**Figure C.1: Client/Server architecture of the Educational Interactive System**

The server program is written in C++. Figure C.2 describes the class diagram of the server program.
Figure C.2: Class Diagram of the server program

Each box indicates a class in the server program which contains a name of the class (bold word) and instances. The upper level classes, which have an outgoing arrow, use objects of the lower level classes which have an incoming arrow. For example, the Chat class has objects of File, Timer, Attend, and Request. The Chat class is the highest level of the class in the program structure and utilizes
data structures and instances of all other classes directly or indirectly. The instances of the Chat class denote well the primary function of the server program. These instances are capable of establishing TCP/IP connection, receiving and sending messages, registering participants, scheduling the requests from participants, setting a timer, and sending screen images.

The Attend class contains a linked list to keep track of the information of all the attendees in a class. When a new participant initiates a session, a participant's node is created and added to the list. When he ends the session, the node will be deleted from the list. The Attend class also has an instance for the calculation of the priority of requests. A structure Adata is used as a node of the list in the Attend class. It contains information including total waiting time, number of opportunities to talk, total amount of talk time, average waiting time per talk, and so on.

The Request class creates a queue structure to maintain incoming requests. The Request class is defined as an object which consists of a five level queue. A node of the queue is defined as a structure Rdata which contains an IP address and the initial of the participant who made a request.

A structure Node is publicly defined to provide a primitive linking capability for the Adata, Rdata, and Queue
class. The Queue class owns instances for the basic manipulation of linking structure.

The File class handles the screen data of the session. It keeps all the screen images during the class and extracts a specific page segment for the request.

The Timer class provides the capability of setting a time limit for a current talker in the class. If the timer is set, the talk session of the current talker in the class will be terminated within certain time limits. The server program is not able to use a system call, sleep() to achieve a timer function, because a server process needs to be always awake to receive a client's request. Therefore, a child process is created (fork) to communicate with the server process by another socket interface. When the time limit comes, the child process sends a message to the server process. The server process receives the message just like the message from clients and reacts as requested.

Particularly, a system call gettimeofday() is used to get the time stamp in a timer function.

The client program of the Educational Interactive System is written in C within a single file. The diagram of the client program routines is shown in Figure C.3 below.
The curses library is used to divide a screen into three windows. The program starts with initialization of the windows by init_win routine. Then the establish routine establishes the connection with the server program. Once the connection is established, the chat routine handles the communication with the server. The program is an event driven execution which waits for input from the keyboard and TCP/IP port. When the program receives a message, it responds as the message requested. Non-blocking capability is used to handle multiplexing I/O that the client gets messages either from user via keyboard or the server through I/O port. A system call, select() provides the capability of handling multiple requests. This system call allows the user process to listen to multiple events, such as keyboard input and the message from I/O port, and to react only when one of
these events occurs. The method of the I/O multiplexing is written in [17].

The client program also has a caching function based on LRU replacement algorithm. Ten screen page size of array is allocated as a cache on its execution so that it resides on the virtual memory space. When user requests a screen page, the check_cache routine checks the local cache first. If the page is in the cache, the prt_routine routine displays it. If it is not in the cache, it will be retrieved from the server. When the cache is full, the lru routine is called to find the page segment for the replacement in the cache.

The Educational Interactive System utilize UDP socket interface. The UDP implementation of the system is simpler than the TCP implementation. The server program needs to receive multiple messages from clients simultaneously. This means that the server with the TCP implementation needs to create multiple processes to make virtual connections with all the clients. Those server processes also need to communicate with each other to make a database of the system. These requirements may complicate the system significantly. With the UDP implementation, however, the server needs to have only one process to receive multiple messages from all the clients.
The Educational Interactive System used only one process for the server except the timer function. Utilization of multiple processes may enhance the capability of the server. In such a case, the processes need to communicate with not only clients' processes but also other processes on the server. Then the design and the implementation of the program increase their complexity remarkably. Handling multiple processes requires a considerable amount of effort to implement.

It also simplifies the program if just one port is used for the communication. Since UDP keeps track of the IP address of the sender of each message, the program is able to identify the destination or original address of the message using just one port.

Note that the actual implementation of the server program did not use a file system to keep screen data of the session. It rather used an array in the local memory system because frequent disk I/O access may lead significant overhead to create synchronization problem dealing with requests from clients through TCP/IP port.
ACRONYMS

TCP : Transmission Control Protocol
UDP : User Datagram Protocol
IP  : Internet Protocol
LRU : Least Recently Used
ISO : International Standard Organization
OSI : Open Systems Interconnect model
API : Application Programming Interface
FCFS : First-Come, First-Served
SJF  : Shortest-Job-First
RR  : Round Robin
LFU : Least Frequently Used
EIS  : Educational Interactive System
REFERENCES


