Individual Assignment 2 (25%)

CSE 4600 (Section 01) – Operating Systems – Spring 2022

Submitted to

Department of Computer Science and Engineering

California State University, San Bernardino, California

by

Spencer Wallace (007463307)

All source code can be found in following github repository

https://github.com/SpencerDWallace/CSE4600/tree/master/Spencer-Wallace-007463307-Assignment2

Due Date: May 15, 2022 11:59 pm

No copy and paste from other colleagues with the same answers and description (particularly in part 1) is allowed. It is required that you carry out the exercises by yourself (with the possibility for collaborating with other colleagues) and provide descriptions (with screenshots wherever necessary) based on your own experience. Copied material from other colleagues will be considered as cheating and dealt with seriously through University academic integrity policies.

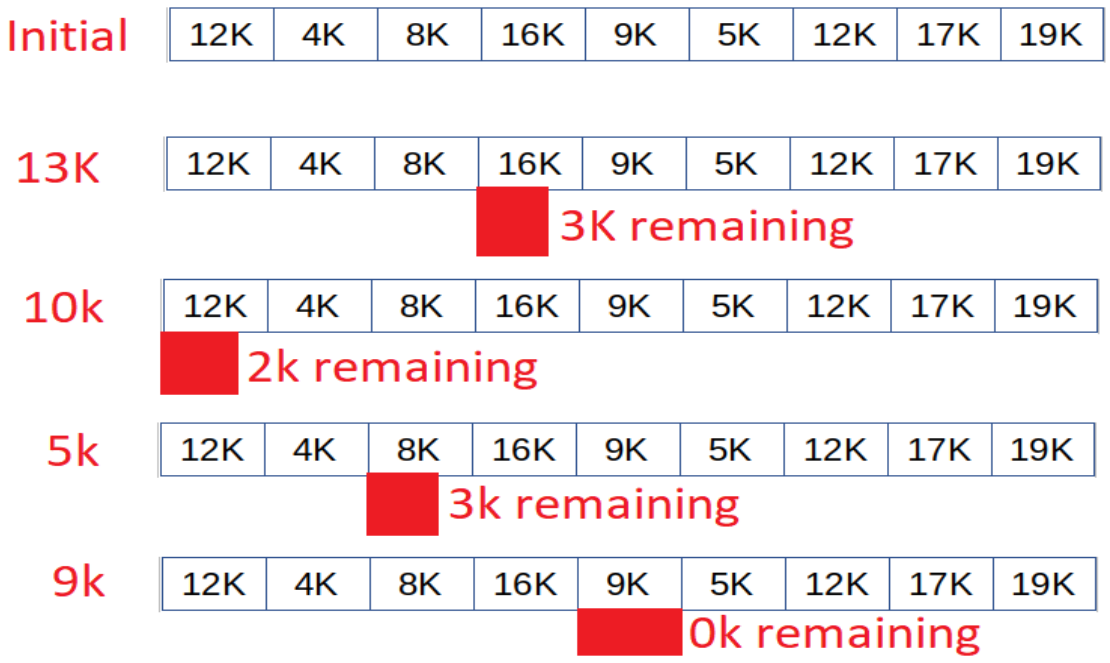
Email: 007463307@coyote.csusb.edu

1. **(5%) Consider the swapping system below with the memory blocks in the following order. Assume that the following segment requests arrive in the order given**
   (a) 13K
   (b) 10K
   (c) 5K
   (d) 9K

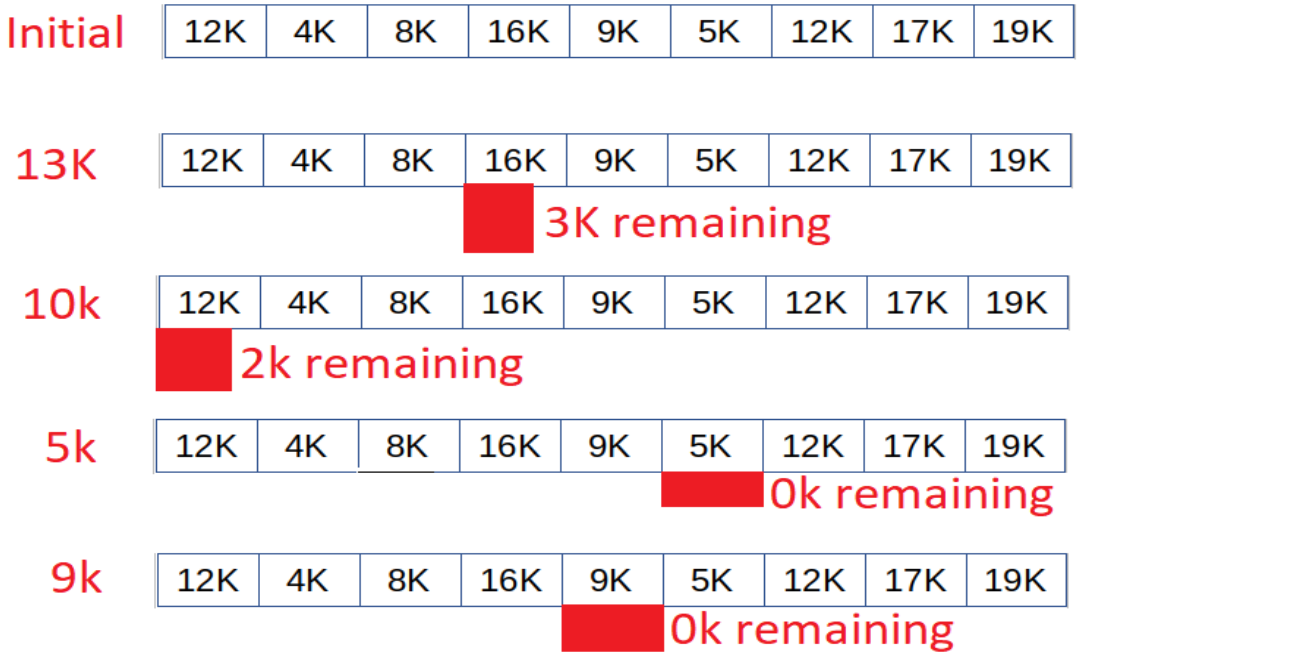
12K	4K	8K	16K	9K	5K	12K	17K	19K

Which blocks will be taken for the above requests for the algorithms we covered in the class. Fit the above requests for (i) first fit, (ii) best fit, and (iii) worst fit. Provide your answers step by step for each of these algorithms. For example, for First fit, create a section in your report where you will have the block sequence with the block highlighted per request. Once the block is selected for a request, also indicate remaining capacity of the blocks on the right (e.g., if the request (a) is 4K and the first fit finds the first block (i.e., 12K), then highlight the block with 12K for option (a) and write your answer 8K as the remaining block capacity.

First fit



Best Fit



Worst fit

2. (5%) Banker's algorithm. Consider the following system with the resources as given in the tables below

Processes	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	1	4	5	7	5	7	1	1	6
P2	3	0	1	5	3	5			
P3	4	2	5	9	2	6			
P4	2	3	3	2	5	5			
P5	4	0	2	4	0	4			

Answer the following questions

1. Calculate and display the Need matrix

   Need matrix

P1	6	1	2
P2	2	3	4
P3	5	0	1
P4	0	2	2
P5	0	0	2

2. Is the system in safe state? If yes, then show at least one safe sequence of the processes. In addition to the sequence, show how the Available (working array) changes as each process terminates.

   P5, new available = (1,1,6) + (4,0,2) = (5,1,8)
   P3, new available = (4,2,5) + (5,1,8) = (9,3,13)
   P4, new available = (2,3,3) + (9,3,13) = (11,6,16)
   P1, new available = (1,4,5) + (11,6,16) = (12,10,21)
   P2, new available = (3,0,1) + (12,10,21) = (15,10,22)

   System is in a safe state because sequence <P5, P3, P4, P1, P2> satisfies safety criteria

3. Suppose P_2 now has the request (3,1,3). Should this request be granted? Why or why not?

The request **would not be granted immediately** because it is requesting more resources than are available. However, following the safe sequence above the request could be granted.

3. (a) 5% - Consider the following reference string

3	2	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3	1	5

Use the algorithms below with the indicated number of frames to calculate the number of faults for each algorithm and number of frames considered. Please note that all the frames are initially empty which will cause a page fault as demonstrated in the class

Algorithm	Number of frames considered			
	3	4	5	7
FIFO	18 faults of 20	17 faults of 20	15 faults of 20	8 faults of 20
LRU	19 faults of 20	17 faults of 20	16 faults of 20	8 faults of 20
Optimal replacement	14 faults of 20	12 faults of 20	10 faults of 20	8 faults of 20

FIFO WITH 3 FRAMES

3	3	3	3	2	4	4	1	6	7	8	2	4	3	1	6	8	4	2	3
	2	2	2	4	1	1	6	7	8	2	4	3	1	6	8	4	2	3	1
		4	4	1	6	6	7	8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	H	F	F	F	F	F	F	F	F	F	F	F	F	F

FIFO WITH 4 FRAMES

3	3	3	3	3	2	2	4	1	6	7	8	2	4	3	1	6	8	4	2
	2	2	2	2	4	4	1	6	7	8	2	4	3	1	6	8	4	2	3
		4	4	4	1	1	6	7	8	2	4	3	1	6	8	4	2	3	1
				1	6	6	7	8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	H	F	F	F	F	F	F	F	F	F	F	F	F	F

FIFO WITH 5 FRAMES

3	3	3	3	3	3	3	2	4	1	6	7	8	2	4	4	3	3	3	1
	2	2	2	2	2	2	4	1	6	7	8	2	4	3	3	1	1	1	6
		4	4	4	4	4	1	6	7	8	2	4	3	1	1	6	6	6	8
				1	1	1	6	7	8	2	4	3	1	6	6	8	8	8	2
					6	6	7	8	2	4	3	1	6	8	8	2	2	2	5
F	F	F	H	F	F	H	F	F	F	F	F	F	F	F	H	F	H	H	F

FIFO WITH 7 FRAMES

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4
		4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	6
					6	6	6	6	6	6	6	6	6	6	6	6	6	6	7
							7	7	7	7	7	7	7	7	7	7	7	7	8
								8	8	8	8	8	8	8	8	8	8	8	5
F	F	F	H	F	F	H	F	F	H	H	H	H	H	H	H	H	H	H	F

LRU WITH 3 FRAMES

3	3	3	3	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3
	2	2	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3	1
		4	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

LRU WITH 4 FRAMES

3	3	3	3	3	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2
	2	2	4	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3
		4	2	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3	1
				1	6	4	7	8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	H	F	F	F	H	F	F	F	F	F	F	F	F	F

LRU WITH 5 FRAMES

3	3	3	3	3	3	3	2	1	6	6	7	8	2	4	3	1	6	8	4
	2	2	4	4	4	2	1	6	4	7	8	2	4	3	1	6	8	4	2
		4	2	2	2	1	6	4	7	8	2	4	3	1	6	8	4	2	3
				1	1	6	4	7	8	2	4	3	1	6	8	4	2	3	1
					6	4	7	8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	H	F	F	F	H	F	F	F	F	H	F	F	F	F

LRU WITH 7 FRAMES

3	3	3	3	3	3	3	3	3	3	3	1	6	7	7	7	7	7	7	6
	2	2	4	4	4	2	2	2	1	1	6	7	8	2	2	3	1	6	8
		4	2	2	2	1	1	1	6	6	7	8	2	4	3	1	6	8	4
				1	1	6	6	6	4	7	8	2	4	3	1	6	8	4	2
					6	4	4	4	7	8	2	4	3	1	6	8	4	2	3
							7	7	8	2	4	3	1	6	8	4	2	3	1
								8	2	4	3	1	6	8	4	2	3	1	5
F	F	F	H	F	F	H	F	F	H	H	H	H	H	H	H	H	H	H	F

OPTIMAL WITH 3 FRAMES

3	3	3	3	2	2	2	2	2	2	2	4	4	4	4	4	8	6	2	3
	2	2	2	4	4	4	4	4	4	4	8	8	8	8	8	6	2	3	1
		4	4	1	6	6	7	8	8	8	3	1	6	6	6	2	3	1	5
F	F	F	H	F	F	H	F	F	H	H	F	F	F	H	H	F	F	F	F

OPTIMAL WITH 4 FRAMES

3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	4	8	6
	2	2	2	2	2	2	2	2	2	2	2	4	4	4	4	4	8	6	3
		4	4	4	4	4	4	4	4	4	4	8	8	8	8	8	6	3	1
				1	6	6	7	8	8	8	8	1	6	6	6	6	3	1	5
F	F	F	H	F	F	H	F	F	H	H	H	F	F	H	H	H	F	F	F

OPTIMAL WITH 5 FRAMES

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	4
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4	6
		4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	6	8
				1	1	1	1	1	1	1	1	1	8	8	8	8	8	8	1
					6	6	7	8	8	8	8	8	6	6	6	6	6	1	5
F	F	F	H	F	F	H	F	F	H	H	H	H	F	H	H	H	H	F	F

OPTIMAL WITH 7 FRAMES

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	4
		4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	1
				1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	6
					6	6	6	6	6	6	6	6	6	6	6	6	6	6	7
							7	7	7	7	7	7	7	7	7	7	7	7	8
								8	8	8	8	8	8	8	8	8	8	8	5
F	F	F	H	F	F	H	F	F	H	H	H	H	H	H	H	H	H	H	F

3. (b) 3% - Following is the source code for FIFO replacement algorithm that uses a queue to save the referenced page numbers, replacing the oldest page when the frames are used up. That is, the first one that goes into the queue will be the first one to be removed. The following is an implementation of the straight FIFO algorithm, where the *R* bit is not used.

```
// fifo.cpp:  First in first out page replacement
// Compile: g++ -o fifo fifo.cpp
// Execute: ./fifo

#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<deque>

using namespace std;

class Cframe {
public:
  int frameNo;    // frame number
  int pageNo;     // page number
  int r;          // reference bit
  Cframe ( int n, int p )       // constructor
  {
     frameNo = n;
     pageNo =  p;  // no page loaded at beginning
     r = 0;
  }
};

deque <Cframe> Q;
int nFaults = 0;
int  page, frame;

int displayMsg()
{
   cout <<"\npage " << page <<" is allocated to frame " << frame;

   return 1;
}

void fault ()
{
  nFaults++;
  cout<<"\nTotal page faults = " << nFaults;
}
```

```
int search ( const deque<Cframe> &q, int p )
{
   int n = q.size();
   for ( int i = 0; i < n; i++ ){
     if ( q[i].pageNo == p )
       return q[i].frameNo;
   }
   return -1;
}

int main()
{
  int maxFrames;
  cout << "\nEnter max. number of frames allowed in main memory: ";
  cin >> maxFrames;

  int n;

  cout << "\nEnter sequence of page requests (-99 to terminate). ";
  while ( true ) {
    cout << "\nNew page : ";
    cin >> page;
    if ( page == -99 )
      break;
    if (( frame = search ( Q, page )) != -1 ) {
      cout << "\npage "<< page << " already in frame " << frame;
    } else {
      n = Q.size();
      if ( n < maxFrames ) {
        Cframe aFrame ( n, page ) ;
        Q.push_back ( aFrame );
        frame = aFrame.frameNo;
        displayMsg ();
      }  else {
      Cframe aFrame = Q.front();
      Q.pop_front();
        aFrame.pageNo = page;
        Q.push_back ( aFrame );
        frame = aFrame.frameNo;
        displayMsg ();
      }
      fault();
     }
  } // while
```

```
  cout << "\nTotal number of faults: " << nFaults << endl;

  return 0;
}
```

The following is a sample input and output of this program:

Enter max. number of frames allowed in main memory: 3

Enter sequence of page requests (-99 to terminate).
New page: 2
page 2 is allocated to frame 0
Total page faults = 1
New page: 3

page 3 is allocated to frame 1
Total page faults = 2
New page: 2

page 2 already in frame 0
New page: -99

Total number of faults: 2

---

i. Compile the above program and run it with the reference string for FIFO that was used in the class. Copy your output as given in the format above.

$ ./fifo

Enter max. number of frames allowed in main memory: 3

Enter sequence of page requests (-99 to terminate).
New page : 0

page 0 is allocated to frame 0
Total page faults = 1
New page : 1

page 1 is allocated to frame 1
Total page faults = 2
New page : 2

page 2 is allocated to frame 2
Total page faults = 3
New page : 3

page 3 is allocated to frame 0
Total page faults = 4
New page : 0

page 0 is allocated to frame 1
Total page faults = 5
New page : 1

page 1 is allocated to frame 2
Total page faults = 6
New page : 4

page 4 is allocated to frame 0
Total page faults = 7
New page : 0

page 0 already in frame 1
New page : 1

page 1 already in frame 2
New page : 2

page 2 is allocated to frame 1
Total page faults = 8
New page : 3

page 3 is allocated to frame 2
Total page faults = 9
New page : 4

page 4 already in frame 0
(exited at this point)

ii. Try the belady's anomaly example that was discussed in the class. Compare the page faults resulted from (i) and show that belady's anomaly occurred.

Belady's anomaly did in fact occur since the number of frames were increased and the number of page faults also increased.

$ ./fifo

Enter max. number of frames allowed in main memory: 4

Enter sequence of page requests (-99 to terminate).
New page : 0

page 0 is allocated to frame 0

Total page faults = 1
New page : 1

page 1 is allocated to frame 1
Total page faults = 2
New page : 2

page 2 is allocated to frame 2
Total page faults = 3
New page : 3

page 3 is allocated to frame 3
Total page faults = 4
New page : 0

page 0 already in frame 0
New page : 1

page 1 already in frame 1
New page : 4

page 4 is allocated to frame 0
Total page faults = 5
New page : 0

page 0 is allocated to frame 1
Total page faults = 6
New page : 1

page 1 is allocated to frame 2
Total page faults = 7
New page : 2

page 2 is allocated to frame 3
Total page faults = 8
New page : 3

page 3 is allocated to frame 0
Total page faults = 9
New page : 4

page 4 is allocated to frame 1
Total page faults = 10

4. (4%) Socket programming multiple client single server implementation

We covered an example of socket programming in week 8 where we implemented and tested a chat application between a single client and a single server. The code examples for both the client and server are available via Blackboard (Week 8 of Programming discussions section).

Requirement: Your task is to extend TCP server file and allow multiple client connections (and chats). At present, the ports are hardcoded in the programs. For ease, you can make port input as part of the command line arguments for the clients so that you can avoid hard coding port numbers for each newly created clients. At the minimum, two clients should be able to connect with the server (you can test more than 2 for robustness). Please upload your TCP server and client .cpp (or .c) files and write your answers with screenshots in your report. Your report can indicate a brief description of what parts were extended in the code to allow multiple clients and what is your output (with the help of screenshots). For multiple client example, please refer to the bulb activity that was uploaded in Week 8 folder.

Following are the copies of the code covered in the class:

Server:

```
// g++ -o tcpServer tcpServer.cpp
// ./tcpServer Server
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 54550
#define SA struct sockaddr

// Function designed for chat between client and server.
void openChat(int sockfd, char *name)
{
    char buff[MAX], clientName[MAX];
    int n;

    // infinite loop for chat
    for (;;) {
        bzero(buff, MAX);

        // read the message from client and copy it in buffer
        read(sockfd, buff, sizeof(buff));
        // print buffer which contains the client contents
        printf("From client: %s \n %s: ", buff, name);
```

```
        bzero(buff, MAX);
        n = 0;
        // copy server message in the buffer
        while ((buff[n++] = getchar()) != '\n')
            ;

        // and send that buffer to client
        write(sockfd, buff, sizeof(buff));
    printf("Message sent. Waiting for client\n");
        // if msg contains "Exit" then server exit and chat ended.
        if (strncmp("exit", buff, 4) == 0) {
            printf("Server Exit...\n");
            break;
        }
    }
}

// Driver function
int main(int arc, char *argv[])
{
    int sockfd, connfd, len;
    struct sockaddr_in servaddr, cli;
    socklen_t addr_size;

    char *name = argv[1];
    // socket create and verification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Binding newly created socket to given IP and verification
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        printf("socket bind failed...\n");
        exit(0);
    }
    else
```

```
        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification
    if ((listen(sockfd, 5)) != 0) {
        printf("Listen failed...\n");
        exit(0);
    }
    else
        printf("Server listening..\n");
    len = sizeof(cli);

    // Accept the data packet from client and verification
    connfd = accept(sockfd, (SA*)&cli, &addr_size);
    if (connfd < 0) {
        printf("server acccept failed...\n");
        exit(0);
    }
    else
        printf("server acccepted the client...\n");

    // Open chat
    openChat(connfd, name);

    // After chatting close the socket
    close(sockfd);
}
```

Client:

```
// g++ -o tcpClient tcpClient.cpp
// ./tcpClient Client
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAX 80
#define PORT 54550
#define SA struct sockaddr

void openChat(int sockfd, char *name)
{
    char buff[MAX], serverName[MAX];
```

```
    int n;


    for (;;) {
        bzero(buff, sizeof(buff));
        printf("%s : ", name);
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        printf("Message sent. Waiting for server\n");
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strncmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main(int arc, char *argv[])
{
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    char *name = argv[1];

    // socket create and varification
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else
        printf("Socket successfully created..\n");
    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);

    // Client to server socket connection
    if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
        printf("Could not connect with server...\n");
        exit(0);
```

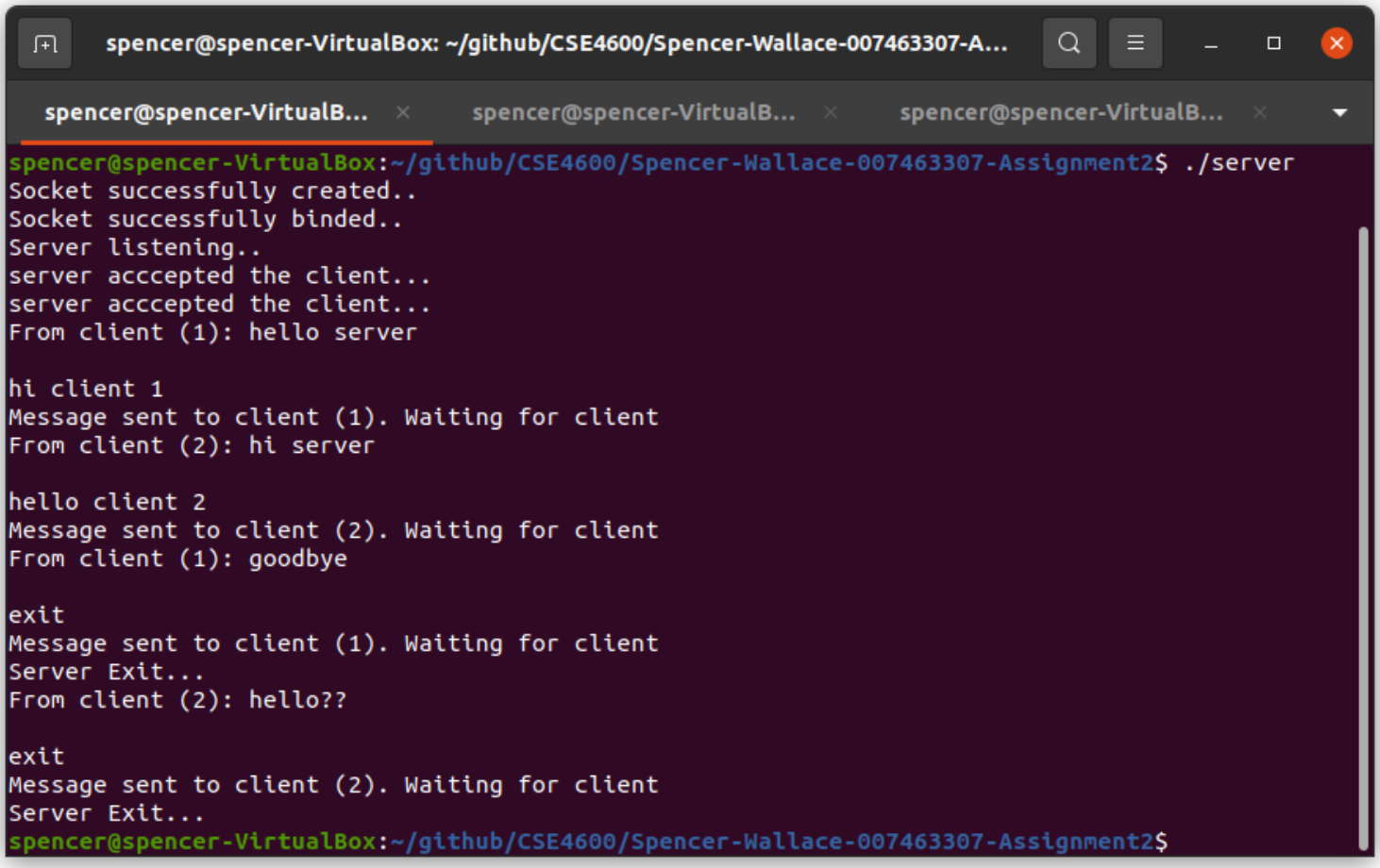
```
    }
    else
        printf("connected to the server..\n");

    // open chat
    openChat(sockfd, name);

    // close socket
    close(sockfd);
}
```

*Sample Output From Server – code on github, server.cpp and client.cpp

- To handle multiple client connections I used the main thread to control accepting connections, and when a new connection was made the main thread would create a new thread to handle the chatting with this new client.



5. (3%) Implement a synchronized call center (*please also see note at the end for bonus*)

One of the typical software engineering questions by industrial sectors is to design or implement a solution of their choice using a programming language of your choice. We will go over a famous example which is normally discussed online very often. Therefore, you will find several places for help when you do this exercise. You are welcome to read from external sources including any help for coding solutions. However, please clearly cite any useful resources you will use for this exercise. DO NOT use the same code if you do find it online. Please submit your solution in the form of a program (Java program NOT recommended) for this exercise along with a paragraph of description on how you implemented your solution. Imagine you have a call center with employees as agents to receive calls. An incoming telephone call must be allocated to an employee who is free. The customers making the phone calls to a call center are referred to as threads. Think of the scenario as a producer/consumer where producers are the employees working in the call center and your critical section is the phone call simulated with thread function. Implement a thread-based solution where you have at least 5 customer threads. The names of the customers should be Alice, Bob, John, Mark, Alex. Please note that when a phone call is made (i.e., the critical section is being executed) the critical section should be secured (either with semaphore or mutex lock).
Your sample output should be something like below

```
Welcome to the call center. Please wait for the setup to start
Initial operator availability = 3
Alice is waiting to speak to the operator...
Bob is waiting to speak to the operator...
John is waiting to speak to the operator...
Mark is waiting to speak to the operator...
Alex is waiting to speak to the operator...
All operators are currently busy assisting other callers - MUSIC ...
Alice is getting the connection to the operator ...
Alex is getting the connection to the operator ...
All operators are currently busy assisting other callers - MUSIC ...
John is getting the connection to the operator ...
All operators are currently busy assisting other callers - MUSIC ...
Alex is now ending the conversation with the operator...
Available operators=1
Mark is getting the connection to the operator ...
John is now ending the conversation with the operator...
Available operators=1
```

For specific help, please refer to the producer/consumer and thread/process synchronization problems covered in various sessions in our course. Please clearly cite the references with links to any supporting material that you will use outside the class content, including any coding sources. Please visit the following links for more help and getting the ideas of implementation

1. Java thread version
2. Javascript non-thread example

NOTE: Successful implementation with a working solution will provide a 3% bonus grade of your remaining total grade (i.e., your final exam will be graded for the remaining grade – 3% and you will be given 3% extra for this exercise, otherwise your final exam will graded as normal)

**Sample output, code on github – call_center.cpp

- For the implementation of this call center I created a semaphore with sem_open (rather than sem_init so that it would be shared in all threads memory) that allowed 3 threads to gain access to the critical region at a time. Inside the critical region each thread would go to sleep for a random amount of time within a given interval. After this critical region the thread would call sem_post releasing its hold on the semaphore and allowing another thread to enter the critical region (or joining the call with an operator). After all threads have joined and finished their call with the operator the main thread exits.