Operating Systems	Instructors: Zaeem Asif, Lehmia Kiran and Umar Suleman
Midterm Exam I	Max Marks: **55 marks**
Fall 2012	Time Allowed: 90 min

Name: Roll no:

Note: Attempt questions in the designated areas only. State your assumptions clearly if required. There are two sections. From section II you have to attempt only 1 fixed question!

SECTION I - (Attempt All)

1. Answer these short questions **(25 marks)**

    a. What is dual mode in CPU and what type of hardware support is needed to implement it?

In order to ensure the proper execution of the OS, we must be able to distinguish between the execution of operating-system code and user-defined . At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

    b. Give two reasons why caches are useful. What problems do they solve? What problems do they cause? if a cache can be made as large as device for which it is caching (for instance, a cache as large as disk), why not make it that large and eliminate the device?

Caches are useful when two or more components need to exchange data, and the components perform transfers at different speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower devices. The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systems where more than one process may be accessing a datum. A component may be eliminated by an equal-sized cache, but only it: (a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

    c. Draw a state diagram to explain *process state model. The diagram should* show the transitions are valid between the states. Also

Name: Roll no:

comment an event that might cause such a transition from each state to the other.

- Five states: New, Ready, Running, Blocked, Exit
- New : A process has been created but has not yet been admitted to the pool of executable processes.
- Ready : Processes that are prepared to run if given an opportunity. That is, they are not waiting on anything except the CPU availability.
- Running: The process that is currently being executed. (Assume single processor for simplicity.)
- Blocked : A process that cannot execute until a specified event such as an IO completion occurs.
- Exit: A process that has been released by OS either after normal termination or after abnormal termination (error).

d. Precisely differentiate Soft vs Hard interrupts

A hardware interrupt causes the processor to save its state of execution via a context switch, and begin execution of an interrupt handler. Software interrupts are usually implemented as instructions in the instruction set, which cause a context switch to an interrupt handler similar to a hardware interrupt.

e. Precisely differentiate User Mode vs Kernel Mode

- Kernel Mode

Name: Roll no:

- In Kernel mode:

the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

- User Mode

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

f. Precisely differentiate Symmetric vs asymmetric Multiprocessing

Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU.
Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes takes among the slaves, and I/O is usually done by the master only. Multiprocessor can save money by not duplicating power supplies, housings, and peripherals. They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software then uniprocessor systems.

g. If in RR scheduling the time quantum is very large such that the biggest CPU burst of any loaded process is less than the time quantum then the scheduling result will be equal to which other scheduling algorithm? Why?

FCFS *algorithm,*

2. There are five processes A to E to run. Their arrival times are 0, 1, 3, 9 and 12 seconds, respectively. Their processing times are 3, 5, 2, 5 and 5 seconds, respectively. What is the turn avg waiting stime using first-come-first-served, shortest-job-first and round-robin (with 1 second quantum) scheduling? Draw Gantt charts and show proper working. [3 + 3.5 + 3.5 = 10 marks]

FCFS:

0 3 8 10
15 20

A	B	C	D	E

Avg waiting time:
0+(3-1)+(8-3)+(10-9)+(15-12)/5
=2.2

Shortest job first(Non-preemptive)

A	C	B	D	E

0 3 5 10
15 20

Avg. waiting time:
0+(3-3)+(5-1)+(10-9)+(15-12)/5
=2.6

Round Robin:
1: A runs alone. 2-3: A and B. 4-6: A, B and C, and A finishes. 7-8: B and C, and
C finishes. 9: B. 10-11: B and D, and B finishes. 12: D. 13-18: D and E, and D
finishes. 19-20: E, and E finishes.

3. (Processes, Context switch, System calls) The CDC 6600 computers could handle up to 10 I/O processes simultaneously on a single CPU using an interesting form of round-robin scheduling called processor sharing. A context switch occurred after each instruction, so instruction 1 came from process 1, instruction 2 came from process 2, etc. The context switching was done by special hardware, and the overhead was zero. If a process needed T seconds to complete in the absence of competition, what is the maximum amount of time it would need if processor sharing were used with N processes? Assume there are less than 10 processes waiting to run. **(10 marks)**

It will need NT seconds.

Name: Roll no:

4. [CS-C students ONLY] Consider the following two threads, to be run concurrently in a shared memory (all variables are shared between the two threads) **(3+4+3 = 10 marks)**

Thread A

```
for (i=0; i<5; i++) {
      x = x + 1;
}
```

Thread B

```
for (j=0; j<5; j++) {
      x = x + 2;
}
```

Assume a single-processor system, that load and store are atomic, that x is initialized to 0 *before either thread starts*, and that x must be loaded into a register before being incremented (and stored back to memory afterwards). The following questions consider the final value of x after both threads have completed.

a. Give a *concise* proof why x≤15 when both threads have completed

Each x=x+1 statement can either do nothing (if erased by Thread B) or increase x by 1.
Each x=x+2 statement can either do nothing (if erased by Thread A) or increase x by 2.
Since there are 5 of each type, and since x starts at 0, x is >= 0 and x<= (5x1)+(5x2)=15

b. Give a *concise* proof why x≠1 when both threads have completed.

Every store into x from either Thread A or B is ¥ 0, and once x becomes ¥ 0, it stays ¥ 0. *The only way for x=1 is for the last x=x+1 statement in Thread A to load a zero and store a one. However, there are at least four stores from Thread A previous to the load for the last statement, meaning that it couldn't have loaded a zero.*

c. Suppose we replace '**x = x+2**' in Thread B with an atomic double increment operation **atomicIncr2(x)** that cannot be preempted while being executed. What are all the possible final values of x? Explain.

Final values are 5, 7, 9, 11, 13, or 15. The x=x+2 statements can be "erased" by being between the load and store of an x=x+1 statement. However, since the x=x+2 statements are atomic, the x=x+1 statements can never be "erased" because the load and store phases of x=x+2 cannot be separated. Thus, our final value is at least 5 (from Thread A) with from 0 to 5 successful updates of x=x+2.

☺ GOOD LUCK ☺

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Operating Systems	Course Code:	CS2006
	Degree Program:	BSCS	Semester:	Spring 2023
	Exam Duration:	60 Minutes	Total Marks:	55
	Paper Date:	28-Feb-2023	Weight	15%
	Section:	ALL Sections except Section 6A	Page(s):	3
	Exam Type:	Mid 1 Exam		

Student : Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions on the given answer sheet. Clearly mention your attempted question no. with the answers on the answer sheet. Avoid unnecessarily explanation.

CLOs	CLO-2	CLO-2	CLO-2	CLO-3	
Questions	Q-1	Q-2	Q-3	Q-4	Total
Total Marks	20	10	10	15	**55**
Marks Obtained					

Question 01: (20 points)

A) What output will be at **Line X** and **Line Y**? (10 points)

```
#include <sys/types.h>
#include<iostream>
#include<unistd.h>
using namespace std;

#define SIZE=5

int nums[SIZE] = {0,1,2,3,4};

int main() {
      int i;
      pid_t pid;
      pid = fork();
      if (pid == 0)
            {
```

```
            for(i=0; i<SIZE; i++) {
            nums[i]*=-i;
            cout<< "CHILD: "<<nums[i]<<endl; // LINE X
            }
      } else if (pid > 0) {
            wait(NULL);
            for(i=0; i<SIZE; i++)
            cout<< "Parent: "<<nums[i]<<endl; // LINE Y
            }
return 0;
}
```

B) Consider the following code: (5+5 = 10 points)

```
if ((fork() || fork()) && fork())
      fork();
printf ("Hello World");
```

How many times **"Hello World"** will be printed on the screen? Also draw the ***Process Tree*** of the given program.

Question 02: (10 points)
What is the context-switch? Explain it with the help of a diagram. Why is context-switch considered as overhead?

Question 03: (10 points)

- **A)** Write a C program that forks a child process that ultimately becomes a zombie process. This zombie process must remain in the system for at least 10 seconds. (4 points)
- **B)** Why do we need **init** process? (2 points)
- **C)** Answer **YES/NO** and provide a brief explanation. (4 points)

(a) Can two processes be concurrently executing the same program executable?

(b) Can two running processes share the complete process image in physical memory (not just parts of it)?

Question 04: (15 points)
The following processes are being scheduled using a **Preemptive, Round-Robin** scheduling algorithm.

Process	Arrival	Burst	Priority
P1	0	20	40
P2	25	25	30
P3	30	25	30
P4	60	15	35
P5	100	10	5
P6	105	10	10

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P_{idle}). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 20 units. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue.

1. Show the scheduling order of the processes using a **Gantt chart**.
2. What is the **Turnaround time** for each process?
3. What is the **Waiting time** for each process?

No rough sheet given on which is not available

235

Operating System (CS-205)

Mid-1, Fall 2016

Date: September 21, 2016 | Marks: 50 | Time: 60 mins

Roll# L13-5846 | Section A

Instructions: *There is no need for a separate answer sheet. Your answer should be concise enough to take only the space given to you. You can ask for a sheet for rough work, where you can first write the answer, amend it sometimes later, but only write on this sheet when you are sure. Cutting on this sheet will reduce your marks.*

Question 1 (5 points): List down the four layers of a computer system.

1. hardware
2. application programs
3. operating system
4. user

5

Question 2 (5 points): We know that it is hard to define the boundaries of an Operating System. Still, there are few components of an operating system which are essential to use a computer hardware properly. You are required to list these *three* components. (Hint: think about the microkernel.)

1. user kernel
2. Application interface
3. Memory

1

Question 3 (10 points): What happens when a user presses a key on the keyboard? List the steps which take the key code from the hardware (keyboard) and hand it over to the application. Each step you write here, you must mention one component of the Operating System responsible for completing that action.

- passes the key code to keyboard buffer (device)
- moves that code from buffer to memory (keyboard buffer)
- generates interrupt (keyboard screwdriver)
- asks for CPU to service the routine (interrupt)
- CPU goes to interrupt vector table
- picks the address of the keyboard routine from I
- services the routine (CPU)
- passes the keycode to application (from memory)
- returns control to CPU
- goes back to the next instruction to be executed

9

Department of Computer Science
National University of Computer & Emerging Sciences, Lahore

Operating System (CS-205)

Mid-1, Fall 2016

Date: September 21, 2016 | Marks: 50 | Time: 60 mins

Question 4 (5 points): Briefly explain the difference between a program and a process.

Program is passive as it resides in secondary storage and can be accessed even when CPU is rebooted whereas
Process is active entity as it resides in primary memory ram and is responsible for getting the process executed init and does not appear after rebooting.

4/

Question 5 (5 points): Define "dispatch latency". What is the effect of scheduling algorithm on "dispatch latency"? Can a scheduling algorithm increase or decrease "dispatch latency"?

dispatch latency. It is basically the time to remove one process and load another process. Such dispatch latency is affected by scheduling algorithm as multiple processes are being removed and loaded. It decreases dispatch latency.

2.5

Question 6 (10 points): List the steps involved in switching two processes.

- saves the state of the previous process in PCB.
- loads the (next) process in its PCB.
- allocate the resources to a process i.e (memory) etc
- brings the process to the main memory (RAM).
- Executes the process
- removes the process from memory after execution
- goes to the saved state of previous process
- gives memory to that process
- executes the process
- restores the state.

2/

OPERATING SYSTEM (CS-205)

Mid-1, Fall-2016

Date: September 21, 2016 | Marks: 50 | Time: 60 mins

Question 7 (5 points): Where does the kernel store the information about a process? List the main elements of that information.

I = Process control block (PCB)
Register
program counter
process priority
process state
pointer

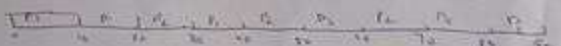
2/

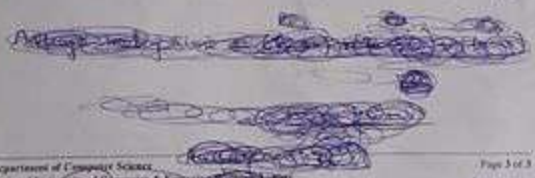
Question 8 (5 points): The first table shows the arrival times of processes. The second table shows which process executes which quantum according to round robin scheduling algorithm. Calculate the average waiting time in this case.

Process Name	Arrival Time	Required Burst Time
P_1	0	10
P_2	20	70
P_3	40	10

Quantum Number (each 10 cycles)	Executing process's name
1	P_1
2	P_2
3	P_3
4	P_2
5	P_2
6	P_2
7	P_2
8	P_2
9	P_2

5/





$P_1 = 10$ (As observed by seeing arrival time and calculated on calculator)
$P_2 = 20$
$P_3 = 10$

Avg waiting time $= \frac{10+20+10}{3} = 13.33$

Operating Systems

Mid #1

Time 1 Hour- Marks 30

Q1. Answer very briefly but clearly the following: (single sentence answers) 5X3=15

a. If the scheduling scheme is preemptive SJF and a process enters the system, on what condition the system will preempt the running job?

b. What is preemptive multitasking?

c. What is the difference between user managed threads and kernel managed threads?

d. If the condition of bounded waiting is not satisfied, what will happen?

e. If ten threads are running within a program and one of them requests I/O and blocks in the kernel, what will happen to the rest of the threads if they are:

   A. Kernel Managed

   B. User Managed.

What will happen if the same task is accomplished using multitasking rather than multithreading?

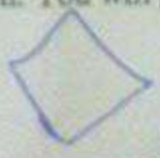
Q2. The following code segments are two independent threads of the same program manipulating a queue `txbuff`. Is there any **critical section** code? Point out and elaborate the operation where mutual exclusion is required and solve this problem using wait(S) and signal(S) TI and SBUF are some global variables which are of no interest at this moment. (15)

```
void tx0(unsigned char c) {
  while ((txhead+1)==txtail);
  txbuff[txhead++]=c;
  txhead%=8;
}
```

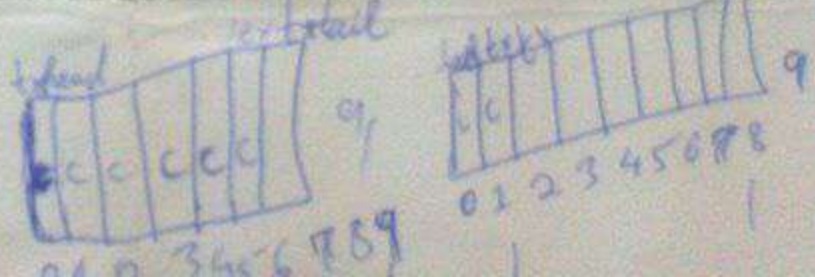
```
void ser0(void) {
 if (TI) {
   TI=0;
     if (txhead!=txtail) {
       SBUF=txbuff[txtail++];
       txtail%=8;
       if (txhead==txtail) {
        txhead=txtail=0;

       }
 }
```

A perfect answer is required in this question. You will get no credit for a partially correct answer.



☺☺☺☺☺☺☺☺GOOD LUCK☺☺☺☺☺☺☺☺



National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	**Operating Systems**	**Course Code:**	**CS 205**
Program:	**Bachelors in Computer Science**	**Semester:**	**Spring 2019**
Duration:	**60 minutes**	**Total Marks:**	**30**
Paper Date:	**26th Feb 2019**	**Weight**	**15**
Section:	**ALL**	**Page(s):**	**3**
Exam Type:	**Mid 1**		

Student : Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Q1: Complete the missing code and answer the Multiple Choice Questions. **[10 marks]**

a) Add the missing code.

The following piece of code—after completion—will senda string from the parent process to the child process. The child process, in turn, will calculate the length of the string and send it back to the parent. The parent will then print the length sent back by the child. The output of the code below should be: (4 marks)

child read Greetings
size sent by child 9

You are required to add missing statements so that the child is able to send the length back to the parent. API for write and read system calls are:

```
ssize_t write(intfd, const void *buf, size_t count);
ssize_t read(intfd, void *buf, size_t count);

//assume all #include statements are there.
#define BUFFER_SIZE 25
#define READ_END        0
#define WRITE_END       1

int main(void) {
        char msg[BUFFER_SIZE] = "Greetings";
        int size;  pid_tpid;
        intfd[2];
        int fd1[2];

        pipe(fd);
        pipe(fd1);
        pid = fork();

        if (pid> 0) {

                close(fd[READ_END]);
                close(fd1[WRITE_END]);

                write(fd[WRITE_END], msg, strlen(msg)+1);
                read(fd1[READ_END], &size, 4);

                printf("size sent by child %d\n", size);

                close(fd[WRITE_END]);
                close(fd1[READ_END]);

        } else {
                close(fd[WRITE_END]);
                close(fd1[READ_END]);

                read(fd[READ_END], msg, BUFFER_SIZE);
                printf("child read %s\n",msg);
                size = strlen(msg);

                write(fd1[WRITE_END], &size, 4);

                close(fd[READ_END]);
                close(fd1[WRITE_END]);

        }
        return 0;
}
```

b) Which of the option arranges the following technologies in the order from fastest to slowest:

a. Hard-disk drives, main memory, cache, registers
b. Registers, main memory, hard-disk drives, cache
c. Registers, cache, main memory, hard-disk drives
d. Cache, registers, main memory, hard-disk drives

c) When two processes communicate through Message-Passing with Zero Capacity buffering, the process sending the message:

a. Gets an error returned to it if the receiver is not ready.
b. Gets blocked if the receiver is not ready.
c. Cannot send a message because the buffer size is zero.
d. Allocates more memory before calling the send() method

d) OS can be defined as a a. Resource Allocator b. Control Program c. User Program d. Both a and b	**e)** How is modularity added to the Linux kernel that is typically a monolithic kernel: a. By adding more system calls b. Through loadable modules c. By adopting micro-kernel approach in Linux d. Both a and b
f) In the five-state model why would a process move from Running to Ready state? a. The process has terminated b. The process needs to perform an I/O operation c. The process' time quantum has expired d. The process needs to execute a system call	**g)** A process stack does not contain: a. Function parameters b. Local variables c. Return addresses d. PID of child process

Q2: CPU Scheduling **[10 marks]**

Suppose four (4) processes given in the table below. You have to execute them using a scheduling algorithm that allows *preemption* and prefers executing the process with the least (minimum) CPU (remaining) bursts. The arrival times and the CPU bursts needed to complete them are also provided. Among the four processes, P_2 needs I/O bursts to complete its execution such that after every 3 CPU bursts (3 time units to be specific), it requires I/O bursts time equivalent to 3 CPU bursts.Keeping in view the following requirement, you are required to find the following:

- Draw the Gantt chart to show how these processes would complete their execution
- Find the waiting time of each process and average waiting time of all the processes

Processes	**CPU Bursts needed**	**Arrival Time**
P_1	*13*	*0*
P_2	*6*	*5*
P_3	*10*	*7*
P_4	*3*	*10*

Answer the following:

What is this algorithm called _____Shortest Remaining Time First_____

Waiting time for P1 _(8-5) + (16-10) = 8__ Waiting time for P2 0 + (13-11) = 2

Waiting time for P3 22-7 = 15 Waiting time for P4 _____0_____

Average waiting time _____8+15+2+0 / 4 = 6.25_____

GANTT CHART BELOW:

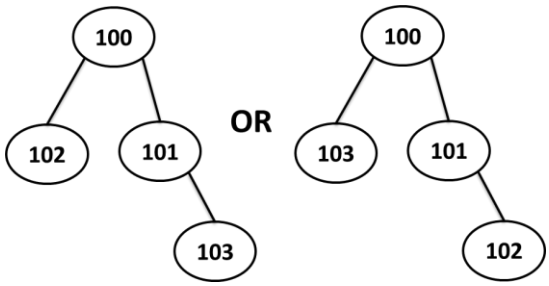
P1	P2	P1	P4	P2	P1	P3
0 5	8	10	13	16	22	32

Q3: Processes **[10 marks]**

Consider the code segment in the right box that creates multiple child processes. Assume a variable NEXT_PROCESS_ID, maintained by the OS, initialized to 100. Each time a new process is created, it gets value of NEXT_PROCESS_ID as its process id. NEXT_PROCESS_ID is then incremented to prepare next id for the next process creation request. There is no compilation or execution error in this code.

1. How many *new* processes are created *(do not count the initial main() process)*?
   Answer: _____**4**_____

2. Create Process tree by showing each process node with its process id. Tree must clearly show parent child relationship for all the processes.



3. Show output of the code below. Write only one sequence if you feel that multiple sequences can be printed.

Parent process waiting for child termination
Message: Welcome
Child Process called
Message: OS course
Child Process called
Message: OS course
Parent Terminating

Green lines are fixed.
Order of yellow lines can be modified. Note that it is not possible for both "Message: OS course" lines to appear above "Child Process Called" lines.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define BUFFER_SIZE 25

int main () {

 char msg[BUFFER_SIZE] = "Welcome";
 pid_t pid = fork ();

 if (pid > 0) {
    strcpy (msg, "Welcome to OS course");
    printf ("Parent process waiting for child termination \n");
    wait (NULL);
    printf ("Parent Terminating \n");
 }
 else {
    printf ("Message: %s \n", msg);
    pid_t pid1 = fork ();

    strcpy (msg, "OS course");
    pid_t pid2 = fork ();

    if (pid2 == 0) {
            strcpy (msg, "Adv OS course");
            printf ("Child Process called \n");
    }
    else  {
            wait (NULL);
            printf ("Message: %s \n", msg);
    }

    if(pid1 > 0) {
            wait(NULL);
    }
 }

 return 0;
}
```

National University of Computer and Emerging Sciences, Lahore Campus

Course Name:	Operating Systems	Course Code:	CS 205
Program:	Bachelors in Computer Science	Semester:	Spring 2019
Duration:	60 minutes	Total Marks:	30
Paper Date:	26th Feb 2019	Weight	15
Section:	ALL	Page(s):	3
Exam Type:	Mid 1		

Student : Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions. Write your name and section clearly in the specified space.

Q1: Complete the missing code and answer the Multiple Choice Questions. **[10 marks]**

a) Add the missing code.

The following piece of code—after completion—will senda string from the parent process to the child process. The child process, in turn, will calculate the length of the string and send it back to the parent. The parent will then print the length sent back by the child. The output of the code below should be: (4 marks)

```
child read Greetings
size sent by child 9
```

You are required to add missing statements so that the child is able to send the length back to the parent. API for write and read system calls are:

```
ssize_t write(intfd, const void *buf, size_t count);
ssize_t read(intfd, void *buf, size_t count);

//assume all #include statements are there and no error in code
#define BUFFER_SIZE 25
#define READ_END        0
#define WRITE_END       1

int main(void) {
        char msg[BUFFER_SIZE] = "Greetings";
        int size;  pid_tpid;
        intfd[2];


        pipe(fd);
        pid = fork();

        if (pid> 0) {

                close(fd[READ_END]);


                write(fd[WRITE_END], msg, strlen(msg)+1);


                printf("size sent by child %d\n", size);

                close(fd[WRITE_END]);


        } else {
                close(fd[WRITE_END]);


                read(fd[READ_END], msg, BUFFER_SIZE);
                printf("child read %s\n",msg);
                size = strlen(msg);



                close(fd[READ_END]);


        }
        return 0;
}
```

b) Which of the option arranges the following technologies in the order from fastest to slowest:

a. Hard-disk drives, main memory, cache, registers
b. Registers, main memory, hard-disk drives, cache
c. Registers, cache, main memory, hard-disk drives
d. Cache, registers, main memory, hard-disk drives

c) When two processes communicate through Message-Passing with Zero Capacity buffering, the process sending the message:

a. Gets an error returned to it if the receiver is not ready.
b. Gets blocked if the receiver is not ready.
c. Cannot send a message because the buffer size is zero.
d. Allocates more memory before calling the send() method

d) OS can be defined as a
   a. Resource Allocator
   b. Control Program
   c. User Program
   d. Both a and b

e) How is modularity added to the Linux kernel that is typically a monolithic kernel:
   a. By adding more system calls
   b. Through loadable modules
   c. By adopting micro-kernel approach in Linux
   d. Both a and b

f) In the five-state model why would a process move from Running to Ready state?
   a. The process has terminated
   b. The process needs to perform an I/O operation
   c. The process' time quantum has expired
   d. The process needs to execute a system call

g) A process stack does not contain:
   a. Function parameters
   b. Local variables
   c. Return addresses
   d. PID of child process

Q2: CPU Scheduling **[10 marks]**

Suppose four (4) processes given in the table below. You have to execute them using a scheduling algorithm that allows *preemption* and prefers executing the process with the least (minimum) CPU (remaining) bursts. The arrival times and the CPU bursts needed to complete them are also provided. Among the four processes, P_2 needs I/O bursts to complete its execution such that after every 3 CPU bursts (3 time units to be specific), it requires I/O bursts time equivalent to 3 CPU bursts.Keeping in view the following requirement, you are required to find the following:

- Draw the Gantt chart to show how these processes would complete their execution
- Find the waiting time of each process and average waiting time of all the processes

Processes	**CPU Bursts needed**	**Arrival Time**
P_1	*13*	*0*
P_2	*6*	*5*
P_3	*10*	*7*
P_4	*3*	*10*

Answer the following:

What is this algorithm called _____

Waiting time for P1 _____ Waiting time for P2 _____

Waiting time for P3 _____ Waiting time for P4 _____

Average waiting time _____

GANTT CHART BELOW: (Draw a neat one!)

Q3: Processes **[10 marks]**

Consider the code segment in the right box that creates multiple child processes. Assume a variable NEXT_PROCESS_ID, maintained by the OS, initialized to 100. Each time a new process is created, it gets value of NEXT_PROCESS_ID as its process id. NEXT_PROCESS_ID is then incremented to prepare next id for the next process creation request. There is no compilation or execution error in this code.

1. How many *new* processes are created *(do not count the initial main() process)*?
   Answer: _____

2. Create Process tree by showing each process node with its process id. Tree must clearly show parent child relationship for all the processes.

3. Show output of the code below. Write only one sequence if you feel that multiple sequences can be printed.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define BUFFER_SIZE 25

int main () {

 char msg[BUFFER_SIZE] = "Welcome";
 pid_t pid = fork ();

 if (pid > 0) {
    strcpy (msg, "Welcome to OS course");
    printf ("Parent process waiting for child termination \n");
    wait (NULL);
    printf ("Parent Terminating \n");
 }
 else {
    printf ("Message: %s \n", msg);
    pid_t pid1 = fork ();

    strcpy (msg, "OS course");
    pid_t pid2 = fork ();

    if (pid2 == 0) {
            strcpy (msg, "Adv OS course");
            printf ("Child Process called \n");
    }
    else  {
            wait (NULL);
            printf ("Message: %s \n", msg);
    }

    if(pid1 > 0) {
            wait(NULL);
    }
 }

 return 0;
}
```

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2017
	Duration:	1 hour	Total Marks:	50
	Paper Date:	18^{th} September, 2017	Weight:	15%
	Section:	All	Page(s):	3
	Exam:	Mid-1	Roll No.	

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, cutting and blotting on this sheet will result in deduction of marks.

Question 1 (2 points): List the missing components of a common computer system, used by a human being.

(1) User

(2)

(3) Operating System

(4)

Question 2 (3 points): List three main resources which an Operating System has to manage.

(a)

(b)

(c)

Question 3 (2 points): Which of the following scheduling algorithms is non-preemptive?

(a) Round Robin

(b) FCFS

(c) Shortest Remaining Time First

(d) Multi Level Feedback Queue

Question 4 (6 points): When you write a piece of code and compile it, you get an output that can be used to accomplish something on a computer system. That output is called a _____(a)_____ and it is stored on _____(b)_____. If we turn of the computer the output will be erased or not? _____(c)_____

(a)

(b)

(c) YES / NO (Circle the right option)

Question 5 (5 points): List down any five elements of a **process control block**.

(1)

(2)

(3)

(4)

(5)

Question 6 (7 points): **Circle** the elements of a process which are shared by threads. **Leave** the elements whose individual copy for each thread is created separately.

(a) Stack Segment

(b) Heap Segment

(c) Code Segment

(d) Data Segment

(e) Open File Pointers

(f) Register Values

(g) Program Counter

Question 7 (5 points): The designer wants an operating system which runs jobs in the order of their arrival. The operating system maintains a queue for this. The operating system does not give CPU to any other job until and unless the running job is completed (terminated).

- Name the **scheduling algorithm** the designer will use in such an operating system
- Can designer use Round Robin Algorithm? give one **reason**

.

Question 8 (5 points): We have studied the life cycle of a process in a common operating system that is described in the form of a state diagram. Draw a **state diagram** for the life cycle of a process, for the scenario mentioned above. Is this state diagram same as we studied in the lectures, or not?

Question 9 (5 points): Suppose you are writing code for an embedded system. Your system will regulate the temperature of a shower. It has one sensor to read the current temperature and an actuator that controls the proportion of hot and cold water. The chip for the controller has the capability of switching between user mode and kernel mode. To implement your solution, is it necessary to use both **user mode and kernel mode**? Give one reason.

Question 10 (5 points): If I use FCFS algorithm in an operating system for personal computers, **what will user experience** if he runs a music player to listen to music while the computer recognizes text from a large amount of big images (a big OCR job). **Will the jobs complete**? yes or no.

Question 11 (5 points): A printer has to print jobs with different priorities. What will be the best **scheduling strategy** that ensures (Hint: use more than one ingredients to build the recipe.)

- All jobs get printed at the end, there is no starvation
- High priority jobs get printed first
- A job is not preempted during its printing, otherwise the output will be garbage.

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2017
	Duration:	1 hour	Total Marks:	50
	Paper Date:	18^{th} September, 2017	Weight:	15%
	Section:	All	Page(s):	3
	Exam:	Mid-1	Roll No.	

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, cutting and blotting on this sheet will result in deduction of marks.

Question 1 (2 points): List the missing components of a common computer system, used by a human being.

(1) User
(2) **Application and System Programs**
(3) Operating System
(4) **Hardware**

Question 2 (3 points): List three main resources which an Operating System has to manage.

(a) **Processes/CPU**
(b) **Memory**
(c) **Permanent Storage**

Question 3 (2 points): Which of the following scheduling algorithms is non-preemptive?

(a) Round Robin
(b) **FCFS**
(c) Shortest Remaining Time First
(d) Multi Level Feedback Queue

Question 4 (6 points): When you write a piece of code and compile it, you get an output that can be used to accomplish something on a computer system. That output is called a _____(a)_____ and it is stored on _____(b)_____. If we turn of the computer the output will be erased or not? _____(c)_____

(a) **Program/Executable**
(b) **Permanent Storage**
(c) YES / **NO** (Circle the right option)

Question 5 (5 points): List down any five elements of a **process control block**.

(1) **Process Number**
(2) **Register Values**
(3) **Memory Limits**
(4) **Open File Pointers**
(5) **Process State**

Question 6 (7 points): **Circle** the elements of a process which are shared by threads. **Leave** the elements whose individual copy for each thread is created separately.

(a) Stack Segment
(b) **Heap Segment**
(c) **Code Segment**
(d) **Data Segment**
(e) **Open File Pointers**
(f) Register Values
(g) Program Counter

Question 7 (5 points): The designer wants an operating system which runs jobs in the order of their arrival. The operating system maintains a queue for this. The operating system does not give CPU to any other job until and unless the running job is completed (terminated).

- Name the scheduling algorithm the designer will use in such an operating system
- Can designer use Round Robin Algorithm? give one **reason**

.

FCFS will be used. Round Robin is preemptive so cannot be used and RR does not fulfill other requirements.

Question 8 (5 points): We have studied the life cycle of a process in a common operating system that is described in the form of a state diagram. Draw a **state diagram** for the life cycle of a process, for the scenario mentioned above. Is this state diagram same as we studied in the lectures, or not?

Only new, ready , running and terminate will be there. There will be no loop back from any state to ready state.

Question 9 (5 points): Suppose you are writing code for an embedded system. Your system will regulate the temperature of a shower. It has one sensor to read the current temperature and an actuator that controls the proportion of hot and cold water. The chip for the controller has the capability of switching between user mode and kernel mode. To implement your solution, is it necessary to use both user mode and kernel mode? Give one reason.

There is no need to use user mode, as there is no user interaction. There is no threat of security breach.

Question 10 (5 points): If I use FCFS algorithm in an operating system for personal computers, what will user experience if he runs a music player to listen to music while the computer recognizes text from a large amount of big images (a big OCR job). **Will the jobs complete**? yes or no.

The music will not be continuous and will only run when the OCR will leave the CPU to fetch images.

Question 11 (5 points): A printer has to print jobs with different priorities. What will be the best **scheduling strategy** that ensures (Hint: use more than one ingredients to build the recipe.)

- All jobs get printed at the end, there is no starvation
- High priority jobs get printed first
- A job is not preempted during its printing, otherwise the output will be garbage.

Multilevel queue is the best strategy to accomplish this, each having a different priority.

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2018
	Duration:	1 hour	Total Marks:	40
	Paper Date:	2nd October, 2018	Weight:	15%
	Section:	All	Page(s):	4
	Exam:	Mid-1	Roll No.	

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, **cutting and blotting on this sheet will result in deduction of marks**.

Question 1 (3 points): List four main components of computer system which the kernel has to manage.

(1)

(2)

(3)

(4) I/O including network management

Question 2 (2 points): The CPU is connected to a _____(a)_____, which is connected to the device controller, which is connected to a _____(b)_____ device. This is the path of communication that enables the hardware interrupts to occur.

(a)

(b)

Question 3 (2 points): Which of the following scheduling algorithms is non-preemptive?

(a) Round Robin

(b) FCFS

(c) Shortest Remaining Time First

Question 4 (2 points): There are machine level commands written inside the CPU which, when executed, can change the values of the registers built inside the device controllers.

(a) True

(b) False

Question 5 (5 points): Write in each cell what type of Inter Process Communication is discussed. Tick the correct column.

Scenario	Shared Memory	Message Passing
Done correctly, sharing of information is faster using this technique		
Processes use `write()` and `read()` system calls		
In some forms only one way communication is possible at one time		
The processes must use some synchronization mechanism		
A queue is used and mostly the queue is controlled by the kernel		

Question 6 (2 points): Suppose a machine runs one instruction in one clock cycle. Now suppose a **program** has 10 instructions in its instruction stream. The program is loaded into memory and becomes a "process". There is no way that the process takes more than 10 clock ticks to finish, is this correct?

(a) Yes

(b) No

Question 7 (4 points): Name any two methods used for parameter passing between a process and the kernel.

(a)

(b)

Question 8 (5 points): Tell the output of the following code. Assume that the parent process running following code has the $PID = 100$. Each new `fork()` creates a new process. Each child process gets the process ID in following way. The first digits of the child process ID are all borrowed from the parent. The last digit is equal to the number of `fork()` done by the parent. For example, if parent whose $PID = 100$, the child created in result of the first fork will have the $PID = 1001$ and the child created in result of the second fork will have $PID = 1002$.
Assume that each instruction runs in the order. Meaning instructions written on smaller line numbers will necessarily execute before the instructions written on bigger line numbers.
Hint:The function `getpid()` returns the *PID* of the calling process.

```
1   # include <stdio.h>
2   int main(void)
3   {
4   int pid=0;
5   pid = fork();
6   if ( pid == 0)
7   {
8           printf("%d,", getpid());
9           pid = fork();
10          if ( pid == 0)
11          {
12                  printf("%d,", getpid());
13                  pid = fork();
14                  if ( pid > 0)
15                  {
16                          printf("%d,", getpid());
17                          pid = fork();
18                          if ( pid > 0)
19                          {
20                                  printf("%d,", getpid());
21                                  pid = fork();
22                                  if ( pid == 0)
23                                  {
24                                          printf("%d,", getpid());
25                                  }
26                          }
27                  }
28          }
29
30  }
31  else if (pid > 0)
32  {
33          printf("%d,", getpid());
34  }
35  return 0;
36  }
```

Question 9 (5 points): Inspired from the above code, write a similar code using `fork()` which prints the string "100,1001,1002,1003" (without the quotes). You can make the same assumption about the execution order as above. Meaning the line written before will execute before.

Question 10 (10 points): Suppose in a machine the CPU executes one instruction per clock cycle. There are three Ethernet cards in the machine. Each machine runs some CPU cycle then reads data from any of the Ethernet cards. The processes arrive in order, i.e. P_1 then P_2 and then P_3

Explanation: The columns CPU Burst+Length show the number of CPU clock cycles needed by the process. The columns I/O Burst+Length show the number of Ethernet cycles needed after each CPU Burst. The table only shows the order in which processes need those cycles, how they will execute depends upon the scheduling algorithm.

- Using the FCFS algorithm list down the order of execution of the processes.
- Calculate the total time of execution of all processes.
- Calculate the average waiting time.

Process Name	Length	CPU Burst	I/O Burst
P_1	3	Yes	-
P_2	6	Yes	-
P_3	8	Yes	-
P_1	12	-	Yes
P_2	4	-	Yes
P_3	7	-	Yes
P_1	7	Yes	-
P_2	5	Yes	-
P_3	3	Yes	-
P_1	13	-	Yes
P_2	10	-	Yes
P_3	7	-	Yes
P_1	3	Yes	-
P_2	25	Yes	-
P_3	12	Yes	-
P_1	17	-	Yes
P_2	15	-	Yes
P_3	8	-	Yes
P_1	3	Yes	-
P_2	3	Yes	-
P_3	3	Yes	-

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-205
	Program:	BS(Computer Science)	Semester:	Fall 2018
	Duration:	1 hour	Total Marks:	40
	Paper Date:	2nd October, 2018	Weight:	15%
	Section:	All	Page(s):	4
	Exam:	Mid-1	Roll No.	

Instructions/Notes: Answer questions on the question paper. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, **cutting and blotting on this sheet will result in deduction of marks**.

Question 1 (3 points): List four main components of computer system which the kernel has to manage.

(1) **CPU management/ Process Management / Process Scheduling**

(2) **Memory Management**

(3) **Permanent Storage Management/ File System**

(4) I/O including network management

Question 2 (2 points): The CPU is connected to a _____(a)_____, which is connected to the device controller, which is connected to a _____(b)_____ device. This is the path of communication that enables the hardware interrupts to occur.

(a) **bus/communication bus/ PCI bus**

(b) **peripheral**

Question 3 (2 points): Which of the following scheduling algorithms is non-preemptive?

(a) Round Robin

(b) **FCFS**

(c) Shortest Remaining Time First

Question 4 (2 points): There are machine level commands written inside the CPU which, when executed, can change the values of the registers built inside the device controllers.

(a) **True**

(b) False

Question 5 (5 points): Write in each cell what type of Inter Process Communication is discussed. Tick the correct column.

Scenario	Shared Memory	Message Passing
Done correctly, sharing of information is faster using this technique	X	
Processes use `write()` and `read()` system calls		X
In some forms only one way communication is possible at one time		X
The processes must use some synchronization mechanism	X	
A queue is used and mostly the queue is controlled by the kernel		X

Question 6 (2 points): Suppose a machine runs one instruction in one clock cycle. Now suppose a **program** has 10 instructions in its instruction stream. The program is loaded into memory and becomes a "process". There is no way that the process takes more than 10 clock ticks to finish, is this correct?

(a) Yes

(b) **No**

Question 7 (4 points): Name any two methods used for parameter passing between a process and the kernel.

(a) **Stack**

(b) **Register / Register and pointer to table**

Question 8 (5 points): Tell the output of the following code. Assume that the parent process running following code has the $PID = 100$. Each new `fork()` creates a new process. Each child process gets the process ID in following way. The first digits of the child process ID are all borrowed from the parent. The last digit is equal to the number of `fork()` done by the parent. For example, if parent whose $PID = 100$, the child created in result of the first fork will have the $PID = 1001$ and the child created in result of the second fork will have $PID = 1002$.
Assume that each instruction runs in the order. Meaning instructions written on smaller line numbers will necessarily execute before the instructions written on bigger line numbers.
Hints:The function `getpid()` returns the *PID* of the calling process. Take due diligence in this question, and cross verify each step. Remember what is the out of `fork()`.

```
1   # include <stdio.h>
2   int main(void)
3   {
4   int pid=0;
5   pid = fork();//(1001)
6   if ( pid == 0)
7   {
8           printf("%d,", getpid()); // 1001
9           pid = fork();//(10011)
10          if ( pid == 0)
11          {
12                  printf("%d,", getpid()); //10011
13                  pid = fork(); //(100111)
14                  if ( pid > 0)
15                  {
16                          printf("%d,", getpid()); //10011
17                          pid = fork(); //(100112)
18                          if ( pid > 0)
19                          {
20                                  printf("%d,", getpid()); //10011
21                                  pid = fork(); //(100113)
22                                  if ( pid == 0)
23                                  {
24                                          printf("%d,", getpid()); //100113
25                                  }
26                          }
27                  }
28          }
29
30  }
31  else if (pid > 0)
32  {
33          printf("%d,", getpid()); //100
34  }
35  return 0;
36  }
```

Solution: 1001,10011,10011,10011,100113,100

Question 9 (5 points): Inspired from the above code, write a similar code using `fork()` which prints the string "100,1001,1002,1003" (without the quotes). You can make the same assumption about the execution order as above. Meaning the line written before will execute before.

```
1   # include <stdio.h>
2   int main(void)
3   {
4           int pid=0;
5           printf("%d,", getpid()); // 100
6           pid = fork();//(1001)
7           if ( pid == 0) // or just printf("%d,", pid);
8           {
9                   printf("%d,", getpid()); // 1001
10          }
11          else if (pid > 0)
12          {
13                  pid = fork();//(1002)
14                  if ( pid == 0) // or just printf("%d,", pid);
15                  {
16                          printf("%d,", getpid()); // 1002
17                  }
18                  else if (pid > 0)
19                  {
20                          pid = fork();//(1003)
21                          if ( pid == 0) // or just printf("%d,", pid);
22                          {
23                                  printf("%d,", getpid()); // 1003
24                          }
25                  }
26          }
27  }
```

Question 10 (10 points): Suppose in a machine the CPU executes one instruction per clock cycle. There are three Ethernet cards in the machine. Each machine runs some CPU cycle then reads data from any of the Ethernet cards. The processes arrive in order, i.e. P_1 then P_2 and then P_3

- Using the FCFS algorithm list down the order of execution of the processes.
- Calculate the total time of execution of all processes.
- Calculate the average waiting time.

Process Name	Length	CPU Burst	I/O Burst
P_1	3	Yes	No
P_2	6	Yes	No
P_3	8	Yes	No
P_1	12	No	Yes
P_2	4	No	Yes
P_3	7	No	Yes
P_1	7	Yes	No
P_2	5	Yes	No
P_3	3	Yes	No
P_1	13	No	Yes
P_2	10	No	Yes
P_3	7	No	Yes
P_1	3	Yes	No
P_2	25	Yes	No
P_3	12	Yes	No
P_1	17	No	Yes
P_2	15	No	Yes
P_3	8	No	Yes
P_1	3	Yes	No
P_2	3	Yes	No
P_3	3	Yes	No

Solutions:

Order of Processes: P_1,P_2,P_3, P_2,P_1,P_3, P_2,P_3,P_1, P_2,P_3,P_1

Total Execution Time: Through the help of Gantt Chart 92

Average Waiting Time: $\frac{0+3+9+7+4+5+27+0+18+0+0+0}{12} = \frac{73}{12}$

National University of Computer and Emerging Sciences, Lahore Campus

	Course:	Operating System	Course Code:	CS-220
	Program:	BS(Computer Science)	Semester:	Fall 2020
	Duration:	1.5 hour	Total Marks:	50
	Paper Date:	17^{th} October, 2020	Weight:	15%
	Section:	ALL	Page(s):	6
	Exam:	Mid-1		

Instructions/Notes: Answer questions on the question paper. Attempt all questions. Programmable calculators are not allowed. Write answers clearly and precisely, if the answers are not easily readable then it will result in deduction of marks. Use extra sheet for rough work, **cutting and blotting on this sheet will result in deduction of marks**.

Name:_____ **Roll No.:**_____ **Section:**_____

Question 1 (4 points): Memory (RAM) is a resource that an OS needs to manage. Allocating certain areas of RAM to a process is one part of managing memory. List two other OS functions for managing RAM.

(1)

(2)

(1) Taking memory back when a process ends.

(2) Protecting one processs RAM from another process.

Question 2 (6 points): How does a process know where the next instruction of the program is? Where is that stored?

Program counter points to the next instruction. Stored in a instruction register.

Question 3 (6 points): Which of the following resources would or would not be shared by multiple threads running in the same process? Write "yes" or "no" at the front, in the empty space. "Yes" means that the resource will be shared.

- Program counter • _____
- Stack memory • _____
- Global variables • _____
- Static data • _____
- Open files • _____

• Program counter	• No
• Stack memory	• No
• Global variables	• Yes
• Static data	• Yes
• Open files	• Yes

Question 4 (12 points): Suppose that you have a program that copies the contents of one source file to a new destination file. The table below shows the steps needed to do this. You have to fill the column which tells what system call will be used for each operation. The first line is already filled for you. Name of the system is not necessary, you can just name the right category. If no system call is used then write NA.

No	Call Sequence	System call interface
1	Acquire input file name (prompt to screen)	I/O operations - `read()`
1	Accept input	
2	Acquire output file name (prompt to screen)	
3	Accept input	
4	Open the input file	
5	If file does not exist, abort	
6	Create output file	
7	If file exists, abort	
8	Loop: Read from input file	
9	Loop: Write to output file	
10	Until read fails	
11	Close output file	
12	Write completion message to screen	
13	Terminate normally	

No	Call Sequence	System call interface
1	Acquire input file name (prompt to screen)	I/O operations `write()`
1	Accept input	I/O operations `read()`
2	Acquire output file name (prompt to screen)	I/O operations `write()`
3	Accept input	I/O operations `read()`
4	Open the input file	File System
5	If file does not exist, abort	Error Detection
6	Create output file	File system
7	If file exists, abort	Error Detection
8	Loop: Read from input file	File system
9	Loop: Write to output file	File system
10	Until read fails	NA
11	Close output file	File system
12	Write completion message to screen	I/O operations `write()`
13	Terminate normally	Program execution

Question 5 (10 points): Based on the table given below use the non-preemptive priority scheduling algorithm and draw a Gantt chart, also calculate the Average Waiting Time.

Process	Burst Time	Priority
P_1	8	5
P_2	6	1
P_3	1	2
P_4	9	3
P_5	3	4

P_2	P_3	P_4	P_5	P_1
0	6	7	16	19

Average Waiting Time = 0+6+7+16+19 = 48/5 = 9.6

Question 6 (12 points): You need to write a program to search an array using concurrent processes. The program searches the array for a given number, and prints the index of the number in the array. If the number is not found then the program shall output -1. You are required to use two child processes to speed up the search. The first child shall search the first half of the array, while the second shall examine the other half. You need not to use any fancy algorithm for the search; rather you can use the simple linear search. Assume no duplicates are there in the array.
Following is the code skeleton:

```
int main() {
        const int N = ...;
        int a[N] = ...;
        int key, i;
        cout << "Enter a number: ";
        cin >> key;

        // Your code to search the array comes here. Write the complete code of the main
            function on the last page.

        cout << "The number found at index: " << i;
        return 0;
}
```

```
int main()
{
        const int N = 10;
        int a[N] = {1,2,3,4,5,6,7,8,9,10};
        int key, i;

        cout << "Enter a number: ";
        cin >> key;

        int pd[2];
        //pipe(pd);
        pid_t pid;
        pid = fork();

                 if(pid < 0)
                 {
                          cout  <<  Error \n  ;
                          return 0
                  }

        if(pid == 0)
        {
                //child 1
                for(int l = 0 ; l < N/2 ; l++)
                {
                        if(key == a[l])
                        {
                                return l;
                        }
                }
                return -1;
        }

                 //parent

        pid1 = fork();
        if(pid1 < 0)
                  {
                            cout <<  Error \n  ;
                            return 0;
                     }
                     if( pid1 == 0)
        {
                //child 2
                for(int l = N/2 ; l < N ; l++)
                {
                        if(key == a[l])
                        {
                                return l;

                        }
                }
                return -1;

        }

    //parent
        int x , y;
    wait(&x)

        if(x == -1)
        {
              wait(&y);
          i = y;
        }
        else
        {
                i  = x
        }

        cout << "The number found at index: " << i << endl;
        return 0;
}
```

National University of Computer and Emerging Sciences

Lahore Campus

036

Operating Systems (CS2006)

Date: September 24th 2024

Course Instructor(s)

Mr. Razi Uddin
Mr. Mubashar Hussain
Ms. Rubab Anam
Ms. Namra Absar

Sessional-I

Total Time (Hrs): 1
Total Marks: 40
Total Questions: 3

22L-7971 Roll No

BSE-5B Section

Student Signature

Do not write below this line

Attempt all the questions.

Follow the order of the questions as given. You must show all relevant work, reasoning, and steps as specified in the question statements to receive full credit.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q1: **[Marks:20]**

You are tasked with writing a C program that demonstrates inter-process communication using pipes. The goal is to create a pipeline between two child processes. The parent should create two child processes. The parent process does not participate in the data transfer between the two children but is responsible for setting up the pipe and properly managing file descriptors. Child1 should execute **Cmd1** and child 2 should execute **Cmd2.**

1. **Cmd1:** `char *cmd1[] = { "/bin/ls", "-a", NULL };`
   // **ls -a** command in Linux lists all the files in the root directory including the hidden files (with dot extension).

2. **Cmd2:** `char *cmd2[] = { "/usr/bin/sort", "-r", NULL };`
   // The **sort -r** command in Linux sorts the lines of text in reverse order.

- **Child 1** redirects its standard output to the writing end of the pipe and executes cmd1 (e.g., **ls**).
- **Child 2** redirects its standard input from the reading end of the pipe and executes cmd2 (e.g., **sort**).
- This means that the **ls** command writes its output to the pipe, and the **sort** command reads its input from the pipe. The **sort** command then sorts the output of the **ls** command in reverse order.
- The parent process closes all unnecessary file descriptors to ensure it does not interfere with the pipe communication.

Requirements:

Write C code with comments explaining the key steps. Provide a step-by-step explanation of how file descriptors are manipulated in both the parent and child processes.

```
//Write your code
char *cmd1[] = { "/bin/ls", "-a", NULL };
char *cmd2[] = { "/usr/bin/sort", "-r", NULL };
int main()
{
//code



exit(0);
}
```

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q2: **[Marks:10]**

Assuming fork() never fails, draw the process tree of the following code, also write how many times "I love FAST", "I love OS", "Breaking", "Exiting" will be printed on screen. (Just write the count).

2 1 4 6

```
int main() {

    while (fork() && fork()) {
        if (fork() || fork()) {
            printf("I love FAST\n");
        } else {
            if (!fork()) {
                printf("I love OS\n");
            }
        }
        printf("Breaking\n");
        break;
    }
    printf("Exiting\n");
    wait(NULL);
    return 0;
}
```

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q3: [Marks: 5]

Consider the following sample code from a simple shell program.

```
int rc1 = fork();
if(rc1 == 0) {
   exec(cmd1);
}
else
{
   int rc2 = fork();
   if(rc2 == 0)
    {
    exec(cmd2);
    }
   else {
    wait();
    wait();
   }
}
```

a. In the code shown above, do the two commands cmd1 and cmd2 execute serially (one after the other) or in parallel (concurrent)?

b. Indicate how you would modify the code above to change the mode of execution from serial to parallel (concurrent) or vice versa. That is, if you answered "serial" in part (a), then you must change the code to execute the commands in parallel (concurrent), and vice versa. Indicate your changes next to the code snippet above.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q4: [Marks: 5]

Which of the following operations by a process will definitely cause the process to move from user mode to kernel mode? Answer yes (if a change in mode happens) or no. (Support your answer with a reason)

a. A process invokes a function in a userspace library.

b. A process invokes the kill system call to send a signal to another process.

National University of Computer and Emerging Sc...

Operating Systems (CS2006)

Date: February 27 2024

Course Instructor(s)

Mr. Razi Uddin

Mr. Mubashar Hussain

Ms. Rubab Anam

Ms. Namra Absar

Sessional-I Exam

Total Time: 1 Hours

Total Marks: 35

Total Questions: 05

Pages: 07

Semester: SP-2024

Campus: Lahore

Dept: Computer Science

Solution

Student Name _____ Roll No _____ Section _____ Student Signature _____

Vetted by _____ Vetter Signature _____

***CLO #: 2* Implement solutions employing concepts of Processes and Threads**

Q1: Imagine a scenario where a parent process interacts with a child process to create a guessing game. Initially, the parent process prompts the user to input an integer number. This number is then transmitted to the child process through a pipe. Subsequently, the child process will compare that number with an already stored secrete number. If the number match, then the child process communicates this information back to the parent process through the pipe. In the case of mismatch, the child process will also inform parent about failure and then the parent process prompts the user again to input a number and transmits it to the child process. This iterative process continues until the child process either successfully guesses the number or exhausts the maximum limit of 5 attempts. Given the skeleton code below you are required to complete the missing portion of the code. **[Marks: 15]**

```
int main() {
int number, tries = 0;
//write your code here
```

int fd[2];
int out[2];

error handling of both

```
while (tries < 5)
{   pid = fork();
    if (pid == -1)
        "error.
}

else if( pid == 0 ){
// Child process
int targetNumber;
int guess=10; //Secrete Number
// write your code here
        close (fd[1]);
        read (fd[0], &targetNumber, sizeof(int));
        close (fd[0]);
        int flag = 0;
if (guess == targetNumber) {
// Notify parent about success (write your code here)
        flag = 1
        write (out[1], &flag, sizeof(int));
        exit(0);
} else {
// Notify parent about the failed attempt (write your code here)
        write (out[1], &flag, sizeof(int));
}
```

```
//write your code here

return 0;



} //Child Process End

else {
// Parent process

 cout << "Enter a number for the child to guess (or enter 'quit' to exit): ";
 string input;
 cin >> input;

  if (input == "quit") {
     break;
  }

  try {
  number = stoi(input);
   } catch (...) {
       cout << "Invalid input. Please enter a valid number or 'quit'." << endl;
       continue;
    }
//write your code here
    write (fd[1], &number, sizeof(int));
    int temp;
    read (out[0], &temp, sizeof(int));
    if (temp){
        // congratsmessage
      } break;

 }//Parent End
    else {
        cout << " could not guess";
        if (tries < 5)
            //try again
        else
            // no more tries
        }
    }
    tries++;
} else { fd[1]); wait (NULL);
return 0;
}
```

***CLO #: 2* Implement solutions employing concepts of Processes and Threads**

Q2: Consider a parent process P that has forked a child process C in the program below. **[Marks:5]**

```
int a = 5;
int fd = open(...) //opening a file
int ret = fork();
if(ret >0) {
close(fd);
a = 6;
...
}
else if(ret==0) {
printf("a=%d\n", a);
read(fd, something);
}
```

After the new process is forked, suppose that the parent process is scheduled first, before the child process. Once the parent resumes after fork, it closes the file descriptor and changes the value of a variable as shown above. Assume that the child process is scheduled for the first time only after the parent completes these two changes.

(a) What is the value of the variable **a** as printed in the child process, when it is scheduled next? Explain.

//Solution

Value changed in parent only.

(b) Will the attempt to read from the file descriptor succeed in the child? Explain.

//Solution

Yes: it will closed by parent

CLO #: 1 Describe services provided by the modern Operating Systems
***CLO #: 2* Implement solutions employing concepts of Processes and Threads**

Q3: Consider a system with two processes P and Q, running a Unix-like operating system as studied in class. Consider the following events that may happen when the OS is concurrently executing P and Q, while also handling interrupts. **[Marks: 5]**

(A) The CPU program counter moves from pointing to kernel code in the kernel mode of process P to kernel code in the kernel mode of process Q.
(B) The CPU stack pointer moves from the kernel stack of P to the kernel stack of Q.
(C) The CPU executing process P moves from user mode of P to kernel mode of P.
(D) The CPU executing process P moves from kernel mode of P to user mode of P.
(E) The CPU executing process Q moves from the kernel mode of Q to the user mode of Q.
(F) The interrupt handling code of the OS is invoked.
(G) The OS scheduler code is invoked.
For each of the two scenarios below, list out the chronological order in which the events above occur. Note that all events need not occur in each question.

(a) A timer interrupt occurs when P is executing. After processing the interrupt, the OS scheduler decides to return to process P.

CFGD.

(b) A timer interrupt occurs when P is executing. After processing the interrupt, the OS scheduler decides to context switch to process Q, and the system ends up in the user mode.

CFGBAE

***CL* *2* Implement solutions employing concepts of Processes and Threads**

Q4: Consider a parent process P that has forked a child process C. Now, P terminates while C is still running. Answer yes/no, and provide a brief explanation. **[Marks: 5]**

(a) Will C immediately become a zombie?

No it will be adopted by init

(b) Will P immediately become a zombie, until reaped by its parent?

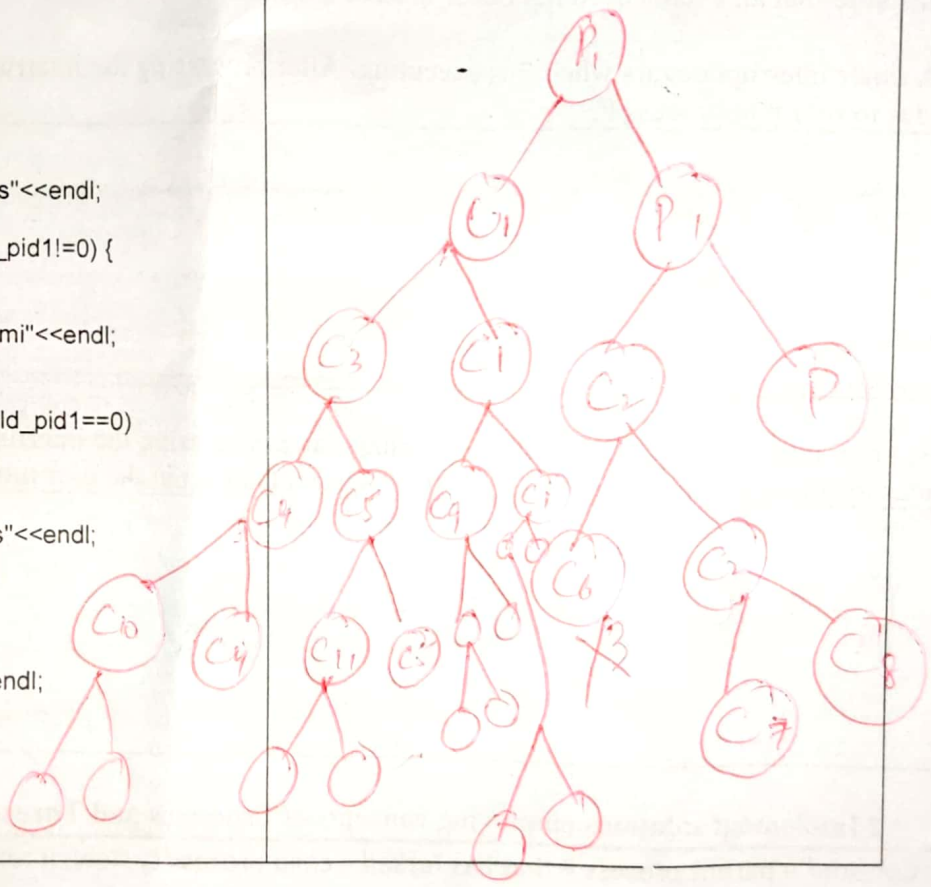
Yes

CLO #: 2 **Implement solutions employing concepts of Processes and Threads**

Q5: Assuming fork() never fails, draw the process tree of the following code, also write how many times "Lahore Qalandars", "Peshawar Zalmi", "Multan Sultans", "Krachi Kings" will be printed on screen. (just write the count). **[Marks: 5]**

```
int main() {
    pid_t child_pid1;
    pid_t child_pid2;
    child_pid1 = fork();
    child_pid2 = fork();
    if (child_pid1 == 0) {
        fork();
        cout<<"Lahore Qalandars"<<endl;
    }
    if (child_pid2 == 0 && child_pid1!=0) {
        if(fork()&& fork())
        {
            cout<<"Peshawar Zalmi"<<endl;
        }
    }
    else if(child_pid2 == 0 || child_pid1==0)
    {
        if( fork() || fork())
        {
            cout<<"Multan Sultans"<<endl;
        }
    }
    else
    {
        cout<<"Krachi Kings"<<endl;
    }
    return 0;
}
```

//Solution



LQ=4

PZ=1

MS=8

KK=1

-16 Total.