Paper Title: Operating System (IOPS332C)

Quiz 1 **Time : 1 hr**

Q1. State True/false with jusification if the answer is false:

a. Multiprogramming (having more programs in RAM simultaneously)
decreases total CPU efficiency (in comparison to Uniprogramming or Batch processing)

A: FALSE. It increases CPU efficiency

b. A forked process shares its parent's data space at all times.
A: FALSE

c. Busy waiting is good as it never allows the cpu to be idle

A: FLASE . It prevents other processes from running and also causes increased power consumption

d. The most commonly used scheduling algorithm for interactive systems is FCFS scheduling
A: FALSE : Round Robin/ prioirity scheduling is prefered in interactive systems

e. Multithreading is useful for application that perform a number of essentially independent tasks that do not be serialized.

A: TRUE

f. In the ``zombie'' state, the process no longer exists but it leaves a
record for its parent process to collect.
A: TRUE

g. If mutual exclusion is not enforced in accessing a critical section,
a deadlock is guaranteed to occur.
A: FALSE.

h. One of the solutions proposed for handling the mutual exclusion problem relies on the knowledge of relative speeds of processes/processors.
A: False. No solution should rely on such assumptions

i. UNIX kernel is designed as a microkernel
A: FALSE. UNIX is a monolithic kernel, meaning that the process,
memory, device, and file managers are all implemented in a single
software module.

j. Context switch means a process switching from a ``blocked state'' to ``ready state''.
A: FALSE.

Q2. Choose the best option for each question :

a. Which component ensures that process can execute within its own address space?
(i) I/O device
(ii) memory addressing hardware
(ii) stack pointers

Ans : ii

b. Which of the following would not necessarily cause a process to be interrupted?

(i) Division by zero
(ii) reference outside user's memory space
(iii) page fault
(iv) accessing cache memory
(v) end of time slice

A: iv

c. At a particular time of computation the value of a counting semaphore is 7.Then 20 P operations and 15 V operations were completed on this semaphore.The resulting value of the semaphore is :

(i) 42
(ii) 2
(iii) 7
(iv) 12

Ans : iii

d. The degree of multi-programming is :

(i) the number of processes executed per unit time
(ii) the number of processes in the ready queue
(iii) the number of processes in the I/O queue
(iv) the number of processes in memory

Ans : (iv)

e. A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero?

(i) This algorithm is equivalent to the first-come-first-serve algorithm
(ii) This algorithm is equivalent to the round-robin algorithm.
(iii) This algorithm is equivalent to the shortest-job-first algorithm..
(iv) This algorithm is equivalent to the shortest-remaining-time-first algorithm

Answer: (ii)

Explanation: The scheduling algorithm works as round robin with quantum time equals to T. After a process’s turn comes and it has executed for T units, its waiting time becomes least and its turn comes again after every other process has got the token for T units.

f. Let the time taken to switch between user and kernel modes of execution be t1 while the time taken to switch between two processes be t2. Which of the following is TRUE?

(i) t1 > t2
(ii) t1 = t2
(iii) t1 < t2
(iv) nothing can be said about the relation between t1 and t2

Ans : iii

Explanation :

Time taken to switch two processes is very large as compared to time taken to switch between kernel and user mode of execution because : Switch between processes or context switc needs saving the PCB of previous process , saving registers and then loading the PCB of new process and its registers etc. However, switch between kernel and user mode of execution, the OS has to just change a single bit at hardware level which is very fast operation.

g. A stack does not contain

(i) function parameters
(ii) local variables
(iii) return addresses
(iv) PID of child process

Ans : iv

h. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

(i) 0%
(ii) 10.6%
(iii) 30.0%
(iv) 89.4%

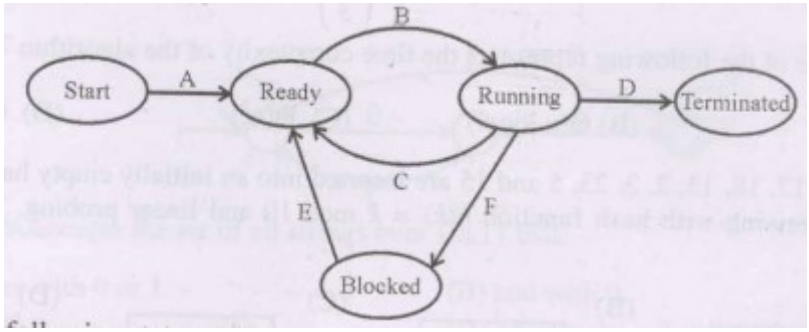
Answer: (ii)

Explanation: Let three processes be p0, p1 and p2. Their execution time is 10, 20 and 30 respectively. p0 spends first 2 time units in I/O, 7 units of CPU time and finally 1 unit in I/O. p1 spends first 4 units in I/O, 14 units of CPU time and finally 2 units in I/O. p2 spends first 6 units in I/O, 21 units of CPU time and finally 3 units in I/O.

```
 idle   p0    p1     p2    idle
 0    2     9       23     44     47
```

Total time spent = 47
Idle time = 2 + 3 = 5
Percentage of idle time = (5/47)*100 = 10.6 %

i. In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state:



Now consider the following statements:

I. If a process makes a transition D, it would result in another process making transition A immediately.
II. A process P2 in blocked state can make transition E while another process P1 is in running state.
III. The OS uses preemptive scheduling.
IV. The OS uses non-preemptive scheduling.

Which of the above statements are TRUE?

(i) I and II
(ii) I and III
(ii) II and III
(iv) II and IV

Ans : (iii) II and III

j. A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution?

(A) -2
(B) -1
(C) 1
(D) 2

Answer: (D)

Explanation: Processes can run in many ways, below is one of the cases in which x attains max value. Semaphore S is initialized to 2

Process W executes S=1, x=1 but it doesn't update the x variable.

Then process Y executes S=0, it decrements x, now x= -2 and
signal semaphore S=1; Now process Z executes s=0, x=-4, signal semaphore S=1
Now process W updates x=1, S=2

Then process X executes X=2

Q3. Q: Given
int X[10];
some appropriate function MyFun
pthread_t Tid;
Write the call to pthread_create(), which executes MyFun, which accepts the array X as an argument
A: pthread_create(&Tid, NULL, MyFun, X)

Q: What is the required function prototype for the function MyFun()?
A: void* MyFun(void* X)

Q: True False A thread may share data space with its parent.
A: TRUE

Q: True False A thread may share code space with its parent.
A: TRUE

Q: True False A thread may share a program counter with its parent.
A: FALSE

Q: True False A thread may share a File Descriptor Table with its parent
A: TRUE

Q: How does a thread acquire RAM that is NOT shared with other threads in the task?
A: Define local variables and use them only in this thread scope.

Q3. (b) Write a C program that will search an array of integers for another given integer.
However, to speedup the search, the search is done in parallel by two child processes.

The parent process reads in the number of integers (max. 100), and then the integers in an array. It also reads in the integer to be searched. It then creates two child processes. The first child process searches the first half of the array, and the second child process searches the second half. If the integer is found, its index in the array is sent to the parent through a pipe. If it is not found, a -1 is sent to the parent through a pipe. The parent waits for both child processes to finish and then prints an appropriate message.

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/ipc.h>
#include <string.h>
void main ()
{
```

```
int i,status,num1,num2,num3,num4, fd1[2], fd2[2];
int a[1000];
char b[5],c[5],d[5],e[5];
pid_t pid1,pid2;

printf("\n\n\nEnter how many numbers - ");
scanf("%d",&num1);
//printf("\n\nEnter the %d numbers below -\n",num1);
for (i=0;i<num1;i++)
{
    printf("%d : ",i);
    scanf("%d",&a[i]);
}
printf("\n\nEnter the number to search - ");
scanf("%d",&num2);
pipe(fd1);
pipe(fd2);
pid1=fork();
if (pid1==0)
{
 printf("this is the child 1\n");
    for (i=0;i<(num1/2);i++)
    {
        if (a[i]==num2)
        {
            printf("found by process 1\n");
            sprintf(b,"%d",i);
            sprintf(c,"%d",-1);
            write(fd1[1],&b,4);
            write(fd2[1],&c,4);
            //kill(0,1);
            break;
        }
        printf("%d\n",a[i]);

    }
    _exit ( EXIT_FAILURE ) ;
}
else
if (pid1>0)
{
    pid2=fork();
    if (pid2==0)
    {
        printf("this is the child 2\n");
        for (i=(num1/2);i<num1;i++)

{
            if (a[i]==num2)
            {
                printf("found by process 2\n");
                sprintf(b,"%d",-1);
                sprintf(c,"%d",i);
                write(fd1[1],&b,4);
                write(fd2[1],&c,4);
                //kill(0,1);
                break;
```

```
            }

            printf("%d\n",a[i]);

        }
        _exit(EXIT_FAILURE);
    }

}

if (waitpid (pid1, &status, 0)>0 && waitpid (pid2, &status, 0)>0)
{

    read(fd1[0],d,4);
    read(fd2[0],e,4);
    num3=atoi(d);
    num4=atoi(e);
    if (num3>0) printf("value of i is %d\n",num3);
    if (num4>0) printf("value of i is %d\n",num4);
}

}
```