Midterm Exam - Introduction to Operating Systems

COP 4610, Fall 2001, Florida State University

Name ....

SSN ....

☐ **COP 4610** ☐ **CGS5765**

☐ **Section 6** (9:05-9:55am) ☐ **Section 7** (10:10-11:00am) ☐ **Section 8** (11:15am-12:05pm)

Score ....

1. **(15 pnts, 3 pnts each) Single Choice -** For each of the following questions, please select the best answer from the given choices. Remember there is only **ONE** best answer to each question.

   (a) The operating system is . . . . [ ]

   i. Application software.
   ii. Hardware.
   iii. System software.
   iv. A user program.

   (b) **Direct memory access** is a technique that . . . . [ ]

   i. Is most effective for slow character devices.
   ii. Cannot be used for block devices.
   iii. Enables the associated controller to read and write data directly from/to primary memory with no CPU intervention during data transfer.
   iv. None of the above.

   (c) In a time-sharing system, an interrupt for a user's program is handled by . . . . [ ]

   i. The user's program directly.
   ii. Operating system for the user program.
   iii. Hardware.
   iv. User's current shell program.

   (d) In a system where round robin is used for CPU scheduling, the following is true when a process cannot finish its computation during its current time quantum . . . . [ ]

   i. The process will terminate itself.
   ii. The process will be terminated by the operating system.
   iii. The process's state will be changed from *running* to *ready*.
   iv. None of the above.

   (e) The following algorithm is proposed to solve the critical section problem between two processes P_1 and P_2, where *lock* is a shared variable.

```
P1                                P2
  do {                              do {
    while (lock) { NULL;}             while(lock) { NULL;}
    lock = TRUE;                      lock = TRUE;
    critical section;                 critical section;
    lock=FALSE;                       lock=FALSE;
    reminder section;                 reminder section;
  } while(TRUE);                    } while(TRUE);
```

   Which of the following statements is true regarding the proposed algorithm? . . . . [ ]

   i. Mutual exclusion to the critical section is guaranteed.
   ii. Both processes can be in their critical section at the same time.
   iii. *Lock* should be initialized to TRUE.
   iv. None of the above.

2. **(20 pnts, 4 pnts each) Multiple Choices -** For each of the following questions, please select all the correct choices from the given ones. Remember there are possibly **MULTIPLE** choices to each question.

   (a) Suppose that a process is executing “counter=counter+1” while another process is executing **concurrently** and **independently** “counter=counter-2”, where the *counter* is a variable shared between the two processes and is accessed only by the two statements. Given that the value of *counter* is **five** before execution, the possible value(s) after both processes finish their statement are . . . . [   ]

   i. Three.
   ii. Four.
   iii. Five
   iv. Six.

   (b) The following statement(s) are true regarding a monitor where no condition variable is defined. . . . . [   ]

   i. Only one process can be running inside the monitor at any time.
   ii. Processes can be blocked inside the monitor.
   iii. No process can be blocked inside the monitor.
   iv. Mutual exclusion is guranteed among all the functions/procedures defined within the monitor.

   (c) The following statement(s) are true regarding system calls . . . . [   ]

   i. System calls are functions defined as part of the operating systems.
   ii. System calls are functions that run in user mode in a dual mode system.
   iii. System calls are implemented using a trap instruction which generates an interrupt.
   iv. System calls are functions that run in system mode in a dual mode system.

   (d) The following functions are necessary for an operating system that supports multiple users and multiple processes. . . . . [   ]

   i. Process management.
   ii. Device management.
   iii. Memory management.
   iv. Non-preemptive scheduling algorithm.

   (e) The following statement(s) are true regarding a device that uses interrupt-driven I/O for I/O operations . . . . [   ]

   i. The CPU must continously check whether the current I/O operation is done.
   ii. The CPU is informed through an interrupt request line when the current I/O operation is done.
   iii. I/O on the device can be overlapped with computation on CPU by other processes.
   iv. A device status table is necessary to save information about pending I/O requests.

3. **(20 pnts, 4 pnts each)** Briefly define and explain the following terms.
   (a) Race condition.

   (b) Context switch.

   (c) Semaphore.

   (d) Direct I/O with polling.

   (e) Process state diagram (Draw a diagram with the three major states and the transitions between states).

4. **(20 pnts)** Assume that the following processes are the only processes in a computer system and that there are no input/output requests from all the given processes. Given the following arrival time and CPU burst time (also called service time) for each process, answer the following questions.

Process	Arrival Time	CPU Burst Time
P_1	0	20
P_2	3	7
P_3	5	2
P_4	7	3

(a) (**15 pnts**) Draw the *Gantt* chart and compute the **average waiting time** for the following CPU scheduling algorithms. Here the waiting time of a process includes all the time it is in the ready list.

i. First Come First Service (FCFS).

ii. **Preemptive** Shortest Job Next (also called Shortest Remaining Job Next).

iii. Round-Robin with time quantum of 4 (Assume that new processes will be inserted at the end of the ready list).

(b) (**5 pnts**) According to your results, which scheduling algorithm among those given in (a) gives the shortest average waiting time? Is that consistent with your theoretical prediction? Justify your answer.

5. (**20 pnts, 5 ptns each**) In a system, there are multiple reader processes which read from a shared file and multiple writer processes which update a shared file. The following variables are shared among all processes:

   int readCount;
   FILE *sharedFile;
   semaphore mutex, writeBlock;

   Reader and writer processes are given in the following C++-like pseudo programs:

Reader Process

```
while(TRUE) {
    <other computing>;
    P(mutex);
    readCount++;
    if (readCount==1)
        P(writeBlock);
    V(mutex);
    Read(sharedFile); //Critical section
    P(mutex);
    readCount−−;
    if (readCount ==0)
        V(writeBlock);
    V(mutex);
}
```

Writer Process

```
while(TRUE) {
    <other computing>;
    P(writeBlock);
    Update(sharedFile); //Critical section
    V(writeBlock);
}
```

(a) How should the shared variable *readCount*, and the semaphores *mutex* and *writeBlock* be initialized so that the system would work properly?

(b) Suppose that when a writer is updating, three new readers have arrived and they are the only reader processes at that time. Show where the readers are blocked.

(c) Suppose that some readers are reading, explain what happens to a newly arrived writer.

(d) Are writers subject to starvation in this solution? Justify your answer.

6. (**20 pnts, 5 pnts each**) Given that each of the following C++ programs compiles and runs successfully without any error. Write the shell command that is equivalent to running each given program once.

   (a) The corresponding shell command is ....

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
#include <sys/wait.h>
int main(void)
{
  char *argv[3];
  int status;
  argv[0] = ``sleep''; argv[1]=``100''; argv[2]=(char *)NULL;
  int pid=fork();
  if (pid == 0)
   execv(``/bin/sleep'', argv);
  wait(&status);
  return 0;
}
```

   (b) The corresponding shell command is ....

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
#include <fcntl.h>
#include <sys/stat.h>
int main(void)
{
  char *argv[2];
  int fd = open(``tmp.out'', O_CREAT|O_TRUNC|O_WRONLY, S_IRWXU);
  close(1); dup(fd); close(fd);
  argv[0] = ``ls''; argv[1]=(char *)NULL;
  execv(``/bin/ls'', argv);
}
```

(c) The corresponding shell command is ....

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
#include <sys/wait.h>
int main(void)
{
  char *argv[3];
  argv[0] = ``sleep''; argv[1]=``100''; argv[2]=(char *)NULL;
  int pid=fork();
  if (pid == 0)
   execv(``/bin/sleep'', argv);
  return 0;
}
```

(d) The corresponding shell command is ....

```
#include <sys/types.h>
#include <unistd.h>
#include <iostream.h>
int main(void)
{
  char *argv[2];
  int fd[2];
  pipe(fd);
  int pid = fork();
  if (pid ==0) {
   argv[0] = ``ls''; argv[1]=(char *)NULL;
   close(1); dup(fd[1]); close(fd[0]); close(fd[1]);
   execv(``/bin/ls'', argv);
  }
  argv[0] = ``cat''; argv[1]=(char *)NULL;
  close(0); dup(fd[0]); close(fd[0]); close(fd[1]);
  execv(``/bin/cat'', argv);
}
```

END OF QUESTIONS