CS 3305: Operating Systems
Department of Computer Science
The University of Western Ontario
Midterm
Spring, 2015

NAME : _____

STUDENT NUMBER : _____

This is a closed book exam. You have 100 minutes to complete 28 questions. *Please write neatly and clearly.* You should have 13 pages.

Question	**Grade**
1-25	____/50
26	____/10
27	____/20
28	____/20
Total	____/100

Score : _____/ 100

Multiple Choice

1. (2 points) An operating system is a program or a group of programs that
   a) helps in checking the spelling of a word
   b) maintains the relationships in a database
   c) manages the resources of the computer
   d) performs the calculations of cells in Excel

2. (2 points) Which of the following resources must be protected by the operating system?
   a) I/O
   b) Memory
   c) CPU
   d) All of the above

3. (2 points) Which is not a function of the operating system?
   a) Memory management
   b) Disk management
   c) Application management
   d) Virus protection

4. (2 points) When a process is created using the *fork()* system call, which of the following is **not** inherited by the child process?
   a) process address space
   b) process identifier
   c) open files
   d) All of the above are inherited

5. (2 points) The text segment of a process address space contains:
   a) the statically allocated data associated with the heap
   b) the dynamically allocated data associated with the process
   c) the executable code associated with the process
   (d) none of the above

6. (2 points) Which of the following is **not** a system call?
   a) Duplicate an open file descriptor
   b) Get the current directory
   c) Display string on screen
   d) Create a new linked list

7. (2 points) ____ scheduling is approximated by predicting the next CPU burst with an exponential average of the measured lengths of previous CPU bursts.
   a) Multilevel queue
   b) Round Robin
   c) First Come First Serve
   d) Shortest Job First

8. (2 points) Among CPU scheduling policies, First Come First Serve (FCFS) is attractive because:
   a) it is simple to implement
   b) it is predictable
   c) it minimizes the total waiting time in the system
   d) it minimizes the average waiting time in the system

9. (2 points) Preemption is when an operating system moves a process between these states:
   a) Running->Ready
   b) Running->Blocked
   c) Ready -> Blocked
   d) Blocked -> Running

10. (2 points) The default scheduling class for a process in Solaris is _____.
    a) time sharing
    b) system
    c) real-time
    d) none of the above

11. (2 points) The goal of a multilevel feedback queue is to:
    a) Keep the priority of interactive processes high
    b) Gradually raise the priority of CPU-intensive processes
    c) Ensure that each process gets the same share of the CPU regardless of how long it runs
    d) Allow the scheduler to provide feedback to the process on how often it is being run

12. (2 points) Which of the following statements are false with regard to the Linux CFS scheduler?
    a) Each task is assigned a proportion of CPU processing time.
    b) Lower numeric values indicate higher relative priorities.
    c) There is a single, system-wide value of *vruntime*.
    c) The scheduler doesn't directly assign priorities.

13. (2 points) The Linux CFS scheduler identifies _____ as the interval of time during which every runnable task should run at least once.
    a) virtual run time
    b) targeted latency
    c) nice value
    d) load balancing

14. (2 points) The two modes of an operating system are called _____
    a) process and kernel
    b) ready and running
    c) interrupt and system
    d) kernel and user

15. (2 points) A thread is
    a) where context switching has low overhead
    b) where context switching has high overhead
    c) where there is no context-switching
    d) none of the above

16. (2 points) A process control block should contain
    a) the process identifier
    b) register values
    c) a list of all open files
    d) all of the above

17. (2 points) Thread-specific data is data that
    a) is not associated with any process
    b) has been modified by the thread but not yet updated to the parent process
    c) is not shared with the parent process
    d) none of the above

18. (2 points) In round-robin scheduling, the time quantum should be _____the context-switch time.
    a) small with respect to
    b) large with respect to
    c) the same size as
    d) irrelevant to

19. (2 points) In scheduling the term *aging* involves
    a) higher priority processes preventing low-priority processes from ever getting the CPU
    b) gradually increasing the priority of a process so that a process will eventually execute
    c) processes that are ready to run but stuck waiting indefinitely for the CPU
    d) processes being stuck in ready queue so long that they are terminated

20. (2 points) Which of the following is not an advantage of multiprogramming?
    a) increased throughput
    b) shorter response time
    c) decreased operating system overhead
    d) ability to assign priorities to jobs

21. (2 points) The disadvantage of round-robin scheduling is that
    a) It gives every process an equal share of the CPU
    b) It can lead to starvation where some processes never get to run
    c) It puts a high priority on interactive processes
    d) It never preempts a process, so a long-running process holds everything up

22. (2 points) A time-decayed exponential average of previous CPU bursts allows a scheduler to
    a) estimate when each process will complete execution and exit
    b) compute the optimum number of processes to have in the run queue
    c) pick the process that will be most likely blocked on I/O the soonest
    d) determine the overall load of the processor

23. (2 points) The process identifier of the first process created when the OS is booted up is
    a) undefined
    b) 0
    c) 1
    d) 3

24. (2 points) System calls:
    a) provide a rich and flexible API for software developers
    b) often change dramatically between different releases of an operating system
    c) are written in Java
    d) none of the above

25. (2 points) In the process state transition diagram, the transition from the READY state to the RUNNING state indicates that
    a) A process was pre-empted by another process
    b) A process has blocked for an I/O operation
    c) A process was just created
    d) A process is done waiting for an I/O operation

26. System Calls (10 points)

a) (4 points) What is the relationship between a trap and a system call handler?

Answer: When a process makes a system call it will end up executing a trap. The trap causes a series of actions to take place . One of these is to pass a system call code to the system call handler. The system call handler uses the code to determine the set of OS instructions that is to be executed.

b) (6 points) Why is a system call table more efficient than if-else statements?

Answer: Let us say that you have 300 system calls. With if-else statements you could evaluation 300 conditions before finding the desired system call. With a system call table you could use the system call code to index into the table (which is implemented as an array).

27. (20 points) Processes, fork, pipes

a) (4 points) How many times does the following program print hello?

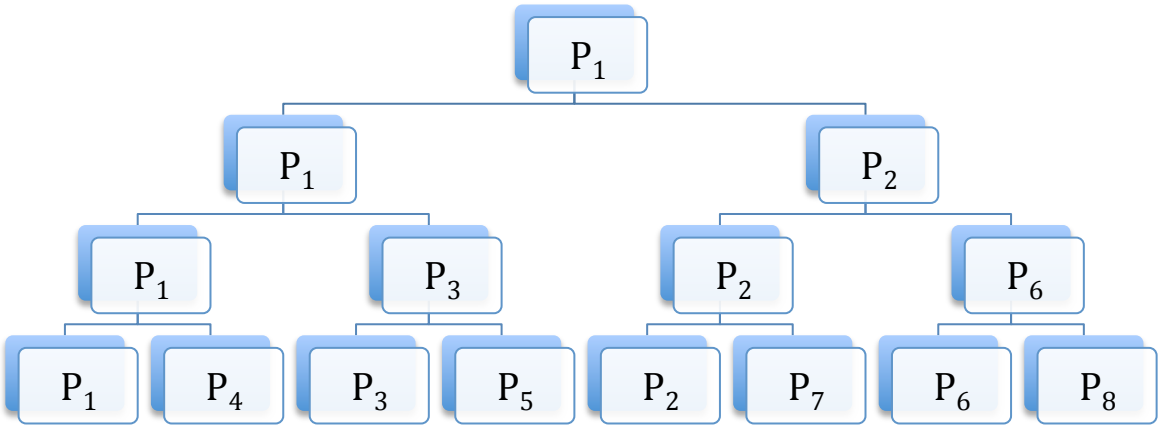
```
int main(void){
   int i;
   for (i=0; i < 3; i++)
     fork();
    printf("hello\n");

}
```

Answer: 8

Let P_1 represent the processes created when the program is started.

When i is 0 P_1 executes the fork – this creates P_2.
When i is 1 P_1 and P_2 execute the fork – this creates P_3 and P_6
When i is 2 P_1, P_2,P_3, P_6 each execute the fork.

The result is that 8 processes are created. This is graphically depicted below.



b) (3 points) Consider the following program. What will the contents of "outfile" be after all processes have successfully exited?

```
int main(void) {
  FILE *fd;
  fd = open("outfile", "w+")
  fork();
  fprintf(fd, "%s", "hello\n");
  exit(0);
}
```

Answer: Two lines of hello

c) (3 points) Consider the following program. What will the contents of "outfile" be after all processes have successfully exited?

```
main() {
   int fd;
   fork();
   fd = open("outfile",  w+)
   write(fd, "hello", 5);
   exit();
}
```

Answer: One line of hello – here the file is opened twice. This means two separate descriptors…

d) (4 points) You write a UNIX shell, but instead of calling *fork()* then *exec()* to launch a new job, you instead insert a subtle difference: the code first calls *exec()* and then calls *fork()* like the following:

```
shell (..) {
 .. ..
 exec (cmd, args);
 fork();
 .. ..
}
```

Answer: Does it work? NO. The "exec" replaces the original code and hence the shell is lost.

e) (4 points) Consider the following program. What should the values be at the question marks on lines A, B, C and D?

```
#include <unistd.h>
#include <stdio.h>
int main(void){
  int n;
  int fd[2];
  pid_t pid;
  char line[80];

  if (pipe(fd) < 0)
       perror("pipe error");

  if ((pid = fork()) < 0) {
        perror("fork error");
   } else if (pid ==  0) {
        close(fd[?]);                                   /*Line A*/
        n = read(?, line, 80)              ;            /* Line B*/
        write(1,line,n)
  } else if (pid > 0) {
        close(fd[?]);                                   /*Line C*/
        n = write(fd[?], "hello world\n")               /*Line D*/
  }
}
```

Line A:_____1_____ Line B: _____fd[0]_____

Line C:_____0_____ Line D: _____1_____

f) (2 points) Consider the following program. How would you fill in the question marks in dup2() if you want the input of "sort" to be from "foobar.txt"?

```
int main(void){
  FILE * fd1;
  char *prog1_argv[4];
```

```
    prog1_argv[0] = "ps";
    prog1_argv[1] = NULL;

    fd1 = open("foobar.txt", "r");
    dup2(?, ?);
    execvp(prog1_argv[0], prog1_argv);
    exit(0);
}
```

Answer: The first question mark should be ___fd1____ and the second question mark should be __STDIN____

28. (20 points) Scheduling

a) (8 points) In a prioritized round robin scheduling policy, new processes are assigned an initial quantum of length q. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice to its current quantum. If a process blocks before its quantum expiries, its new quantum is reset to q. You may assume that a process requires a finite total amount of CPU time.

- Suppose that the scheduler gives higher priority to processes that have larger quanta. Is starvation possible? Why or why not?

  Answer: No. Keep in mind that all processes (regardless if they are I/O bound or CPU bound) are assigned the same quantum and hence are assigned to the same queue. A CPU bound process that completes its quantum is assigned a higher priority but since a process terminates the worst that happens is that the CPU bound process will continue to execute until it is done. Now let us assume that there is an I/O bound process and a CPU bound process arrives after the I/O bound process. The I/O bound process will run before this CPU bound process. The second CPU bound process will run and if we assume that it completes the quantum it will be assigned a higher priority. When that process completes the I/O bound process gets another turn.

  Eventually the I/O bound process will always make it to the head of the low priority queue even if there is a steady stream of CPU bound processes – the reason being that initially a CPU bound process is put in the same queue as the I/O bound process.

- Suppose that the scheduler gives higher priority to processes that have smaller quanta. Is starvation possible? Why or why not?

  Answer: Yes. Suppose a CPU bound process runs and uses its entire quantum. Its quantum is doubled. Suppose a steady stream of I/O bound processes enter the system. With their lower quantum they will always be selected for execution and hence starve the original process.

b) (4 points) The following table consists of a set of jobs to be processed on a single CPU. If a round-robin scheduling algorithm with a quantum value of 2 milliseconds is assumed, then at what time is job 4 completed? You may assume that context switching takes no time.

Job	**Burst**	**Arrival**
1	4	0
2	6	1
3	3	5
4	2	6

Time	Job arrived	Ready Queue	Job Running
0	1		1
1	2	2	1
2		1	2
3		1	2
4		2	1
5	3	2 3	1 Done
6	4	3 4	2
7		3 4	2
8		4 2	3
9		4 2	3
10		2 3	4
11		2 3	4 Done

Some of you started at time 1 and some of you assumed that 4 finished at time 11 or 12 (both were accepted).

c) (8 points) A scheduler uses multi-level feedback queues with the following characteristics

Level	Policy	Time-Slice
1	Round Robin	10 msec
2	Round Robin	40 msec
3	FCFS	Unlimited

Newly created processes are placed in the level 1 queue and move down a level when they completely use their quantum. Processes do not move up levels. All processes at level, i, are selected before processes at levels greater than i. Preemption only occurs when the quantum is completed.

The following table is a list of processes that shows CPU burst times. The processes arrive in the order shown.

Process name	CPU burst time
A	8
B	300
C	60
D	12
E	5
F	20

Using the information above complete the following table showing the order that processes are scheduled, the queue that the process was selected from and how long each process runs. Some of the table has been filled.

Note: The size of the table is larger than it has to be.

Process	Queue	Run-Time
A	1	8
B	1	10
C	1	10
D	1	10
E	**1**	**5**
F	**1**	**10**
B	**2**	**40**
C	**2**	**40**
D	**2**	**2**
F	**2**	**10**
B	**3**	**250**
C	**3**	**10**

This page left intentionally blank