Operating systems
Final examination
(part 1)
2009/2010

Student Name:

NIA:

1. (2pt) Suppose two processes enter the ready queue with the following properties: Process 1 has a total of 8 units of work to perform, but after every 2 units of work, it must perform 1 unit of I/O (so the minimum completion time of this process is 12 units). Assume that there is no work to be done following the last I/O operation. Process 2 has a total of 20 units of work to perform. This process arrives just behind P1. Show the resulting schedule for the shortest-job-first (preemptive) and the round-robin algorithms. Assume a time slice of 4 units for RR. What is the completion time of each process under each algorithm?

Answer:

SJF:

Start Time 0 2 3 5 6 8 9 11 28

Process P1 P2 P1 P2 P1 P2 P1 P2

P1 completes (with I/O) at time unit 12. P2 completes at 28.

RR:

Start Time 0 2 6 8 12 14 18 20 28

Process P1 P2 P1 P2 P1 P2 P1 P2

P1 completes (with I/O) at time unit 21. P2 completes at 28.

2. Explain what will happen when the following C code is executed. Indicate how many times each of the letters “a", “b", “c" and “d” will be displayed and why. Explain your reasoning with the help of a tree diagram. *Also,* show two of the many distinct orders in which all of the letters might be displayed, depending on preemption/scheduling issues. Defend each with an explanation. (Note: assume that if a parent process finishes before a child process, this will not affect the child process, which is sometimes true and sometimes false depending on the operating system's setup.)

```
int i, status;
for (i = 0; i < 3; i++)
{
        if (fork() != 0)
    {
                printf("a");
                if (fork() == 0) {
                        printf("c");
                } else {
```

```
                printf("d");
                exit(0);
            }
        } else {
            printf("b");
        }
}
```

Seven a's, seven b's, seven c's, seven d's will be displayed.

3. Consider the following C code that creates and joins with two threads. Assuming that the threads are scheduled completely before the parent process (i.e., have a higher priority), what will be the output from running this program? Be careful! There is a significant trick!

```
int a = 0;

void *print_fn(void *ptr)
{
  int tid = *(int *)ptr;
  int b = 0;

  a++; b++;
  printf("id: %d a: %d b: %d\n", tid, a, b);

  while (1); // Spin-wait here forever
}

int main()
{
  pthread_t t1, t2;
  int tid1 = 1;
  int tid2 = 2;
  int ret1, ret2;

  a++;
  printf("Parent says a: %d\n", a);
  ret1 = pthread_create(&t1, NULL, print_fn, (void *)&tid1);
  ret2 = pthread_create(&t2, NULL, print_fn, (void *)&tid2);

  if (ret1 || ret2) {
    fprintf(stderr, "ERROR: pthread_create failed\n");
    exit(1);
  }

  if (pthread_join(t1, NULL)) {
```

```
    perror("join of t1");
    exit(1);
  }
  if (pthread_join(t2, NULL)) {
    perror("join of t2");
    exit(1);
  }

  printf("Thread 1 and 2 complete\n");
}
```

Answer:
Parent says a: 1

id: 1 a: 2 b: 1

id: 2 a: 3 b: 1

4. (4pt) True/False

a. Threads are cheaper to create than processes
b. Kernel-scheduled threads are cheaper to create than user-level threads
c. A blocking kernel-scheduled thread blocks all threads in the process
d. Threads are cheaper to context switch than processes
e. A blocking user-level thread blocks the process
f. All kernel-scheduled threads of a process share the same virtual address space
g. The operating system is not responsible for resource allocation between competing processes
h. System calls do not change to privilege mode of the processor

Answers:
a. Threads are cheaper to create than processes – True
b. Kernel-scheduled threads are cheaper to create than user-level threads – False
c. A blocking kernel-scheduled thread blocks all threads in the process – False. This is true for user level threads
d. Threads are cheaper to context switch than processes – True – don't have to save the address space
e. A blocking user-level thread blocks the process - True
f. All kernel-scheduled threads of a process share the same virtual address space – True
g. The operating system is not responsible for resource allocation between competing processes – False – it is responsible for this
h. System calls do not change to privilege mode of the processor – False – we trap into the kernel so we do change the privilege mode