

Name_____ ITSUNIX user name_____

1:_____/25 2:_____/15 3:_____/30 4:_____/25 5:_____/5 Total:_____/100

Univ. at Albany, SUNY CSI 400 Operating Systems Fall 2003 Professor Chaiken
Midterm October 10

This exam is open book and notes. There are 5 parts for a total of 100 points. Answer the specified ones in a blue-book and the rest on the question sheets. Incomprehensible answers get zero points! Answers will be graded on informativeness and on how much understanding they evidence, not just on “legal” correctness.

Part 1 (25 points)-answer in the blue-book

The register, program counter, and PSW (which carries flags to indicate if the most previous test or arithmetic instruction had a positive, negative, zero, overflowed, and/or carry result) must be saved whenever a thread is interrupted, blocked, or otherwise made to stop temporarily after it had been running.

1. (5 points) Why?
2. (5 points) Explain in English, generic assembly language, or both, what the code to save the registers does. It must clearly indicate the direction of data copy operations (to/from registers, to/from RAM memory).
3. (5 points) Register values for “kernel (level) threads” are saved in a different place compared to the place where register values for “user threads” are stored. What is the difference between these places?
4. (5 points) What does the scheduler do during a blocking system call? Say more than “it runs”.
5. (5 points) Why is a hardware timer that produces periodic interrupts necessary for the operating system to implement a scheduling strategy that is pre-emptive (rather than cooperative)?

Part 2 (15 points)-Answer on the next question sheet

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    unsigned char buffer[1000];
    int ret;
    int X;

    ret=read( 0, buffer, 1);
    X = buffer[0];
    printf("%d:%d\n", ret, X);

    //The std. C++ equivalent of the above is:
    // cout << ret << ":" << X << endl;

    ret = read( 0, buffer, 2);
    X = buffer[0];
    printf("%d:%d\n", ret, X);

    ret = read( 0, buffer, 4);
    X = buffer[0];
    printf("%d:%d\n", ret, X);

    ret = read( 0, buffer, 8);
    X = buffer[0];
    printf("%d:%d\n", ret, X);

    ret = read( 0, buffer, 16);
    X = buffer[0];
    printf("%d:%d\n", ret, X);

    printf("Glad that is done!\n");
    //cout << "Glad that is done!" << endl;
    return 0;
}
```

Assume this program in file `blocker.c` is compiled and run once in an common, time-shared POSIX system like `itsunix`, when the following user interaction with the shell and program occurs: Each line of user input is terminated by the “enter” key, as usual.

`% gcc blocker.c` (Compiles/links to generate `a.out`)

`% a.out` (This command runs executable file `a.out`)

`ABCDEFGH` (After a moment of thought, the user types these 8 letters and then presses “enter”.)

`1:65` (First line of process output, really.)

You figure out the rest! Assume no more input is typed by the user.

Answer the following questions about the process described above:

1. What are the other lines printed by the process?
2. How many times does a `read()` system call block during the process? Explain the circumstances for each of these times a call to `read()` blocks.
3. How would this process finally finish?

Part 3 (30 points)-write each of 3 sections on a SEPARATE bluebook page

Consider the following set of processes, each with amount of CPU time in milliseconds it

	P1	8
	P2	2
would use if it ran on the CPU by itself:	P3	3
	P4	1
	P5	5

The processes are assumed to have arrived in the order P1, P2, P3, P4, P5, almost all at time 0.

- A (a) Draw a Gantt chart that illustrates the execution of these processes using ideal, Shortest Job First (SJF) scheduling.
- (b) Calculate the turnaround time and waiting time for each process.
- (c) Calculate the average turnaround time and waiting time for all for the whole execution.

Assume now: Just after process P1 consumes its first 1 millisecond of CPU time, it performs a “sleep” system call with a sleep time of 6 milliseconds. In effect, this system call will block until 6 milliseconds (on the real time clock) have elapsed. After these 6 milliseconds have elapsed, process P1 will be “awakened” and it will then become ready. Answer the next two questions under this new assumption.

- B. Using the non-preemptive First Come First Service scheduling algorithm (in which the awakened process will be placed at the end of the ready queue),
- (a) Draw a Gantt chart that illustrates the execution of these processes;
- (b) Calculate the turnaround times and waiting times for each process;
- (c) Calculate the average turnaround time and waiting time for the whole execution.
- C. Repeated the above 3 questions using the Round Robin (preemptive) scheduling algorithm with quantum=1 millisecond.

Base your answers on the definitions below, quoted from AOSC:

“Turnaround time: From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.”

“Waiting time: The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.”

Part 4 (25 points)-Answer on TWO separate bluebook pages.

```
public class Counter
{
    private int count;
    public Counter() {
        count = 0;
    }
    public void Up() {
        count = count + 1;
        System.out.println("Counter upped to " + count );
    }
    public void Down() {
        count = count - 1;
        System.out.println("Counter downed to " + count );
    }
}
```

Write a Java application in which the main method:

1. Constructs one object of class `Counter`, that will be a shared object (variable) among two Java threads described below.
2. Constructs one object of class `MyThread`, whose methods can access the `Counter` object constructed above.
 - (Your answer should also include a class definition for the `MyThread` class. This class extends the Java language class `Thread`. It should have a `run()` method, so that when a Java Thread of this class is started, the newly started Java Thread repeatedly calls, in an infinite loop, the `Down()` method on the shared `Counter` object. Write this part of your answer in a separate place!)
3. The main method will next start the Java Thread it had just constructed.
4. The main method will finally cause the “main” thread to repeatedly call the `Up()` method of the shared `Counter` object.

In brief, the required Java application will run two Java threads concurrently: The “main” thread will repeatedly increment a shared counter and the second thread will repeatedly decrement the same (shared) counter.

The name of the top level class (which would contain the `main` method) should be `Part4` and the name of the class that extends `Thread`, so it contains the method that the second thread runs, should be `MyThread`.

Use separate pages in the blue-book for the separate “files” `Part4.java` and `MyThread.java`! Don't bother copying my code, which you can assume will be in a file named `Counter.java`!

Part 5 (5 points)-Write your answer BELOW.

Suppose the concurrent execution of two threads with a shared integer variable `COUNT` (like in the previous example) is modelled so that the *atomic operations* are `regA=COUNT`, `regB=COUNT`, `COUNT=regA+1` and `COUNT=regB-1`. Assume `regA` and `regB` are separate “registers”.

Assume the initial value in `COUNT` is 0.

Assume the programs followed by the threads are

ThreadA:

(A1) `regA=COUNT`;

(A2) `COUNT=regA+1`;

ThreadB:

(B1) `regB=COUNT`;

(B2) `COUNT=regB-1`;

There are $6 = \binom{4}{2}$ interleavings of these atomic operations; they represent all possible variations of concurrent execution of these threads under the given model. List (1) all 6 interleavings; and for each, (2) show what happens **each** time the value of `COUNT` changes, (3) show the final value of `COUNT`, and (4) indicate for which interleavings the result is wrong because of the bug caused by races.