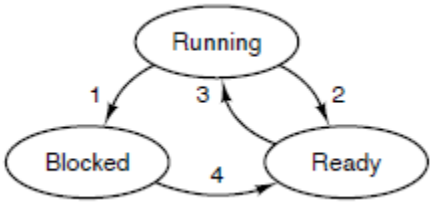
Homework #2
Processes and Threads

1. In the figure below three process states are shown but practically there are a couple more. Briefly describe the other possible states along with their transitions. (4 pts)



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

2. Modern computer systems rely on interrupt handling to respond to hardware and software events. Explain why portions of an interrupt service routine (ISR) are often implemented using low-level assembly instructions rather than entirely in a high-level language. In your answer, discuss what specific processor or system state aspects require this choice. (4 pts)

3. Explain the role of the *init* process on UNIX and Linux systems in regard to process termination.(5 pts)

4. Consider a multi-programmed system with a degree of 6 (i.e., six programs are loaded into memory at once). Each process spends 35% of its execution time waiting for I/O. What is the expected CPU utilization under these conditions? (3 pts)

5. Multiple jobs can execute concurrently and finish sooner than if run sequentially. Suppose two jobs, each requiring 12 minutes of CPU time, start at the same moment. How long will the second job take to finish if they run sequentially? How long if they run in parallel? Assume each job spends 50% of its time waiting for I/O. (4 pts)

6. Modern web browsers must handle many tasks at once. Describe how applying a multithreaded design can improve a browser's responsiveness and overall performance. (3 pts)

7. In Figure 2-11 of the textbook the register set is associated with each thread rather than with the overall process. Explain the reason behind this design choice. Why must each thread maintain its own register context even though the physical CPU provides only a single set of hardware registers? (4 pts)

8. What is the biggest advantage and main drawback of managing threads entirely in user space, especially with respect to OS scheduling and blocking? (4 pts)

9. Suppose that a program has two threads, each executing the `get_account()` function shown below. Identify a race condition in this code. (3 pts)

```
int accounts[LIMIT]; int account_count = 0;

void *get_account(void *tid) {
  char *lineptr = NULL;
  size_t len = 0;

  while (account_count < LIMIT)
  {
     // Read user input from terminal and store it in lineptr
     getline(&lineptr, &len, stdin);

     // Convert user input to integer
     // Assume user entered valid integer value
     int entered_account = atoi(lineptr);

     accounts[account_count] = entered_account;
     account_count++;
  }
```

10. When threads are implemented in user space versus the kernel space, is there a separate stack for each thread or just one per process? (4 pts)

11. Is it possible to have concurrency but not parallelism? Explain. (4 pts)

12. Consider the following code segment: (4 pts)

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
   fork();
   thread_create( . . .);
}
fork();
```

    a. How many unique processes are created?

    b. How many unique threads are created?

13. What is priority inversion in a real-time system? Discuss what solution(s) could be implemented to resolve priority inversion. (4 pts)

14. Assume that a system has multiple processing cores. For each of the following scenarios, describe which is a better locking mechanism — a spinlock or a mutex lock where waiting processes sleep while waiting for the lock to become available: (3 pts)

    a. The lock is to be held for a short duration.

    b. The lock is to be held for a long duration.

    c. The thread may be put to sleep while holding the lock.

15. Show how to implement the `wait()` and `signal()` semaphore operations in uniprocessor environment using busy waiting. (7 pts)

16. Five batch jobs arrive at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the average process turnaround time. Ignore process switching overhead. (8 pts)

    (a) Round robin.
    (b) Priority scheduling.
    (c) First-come, first-served (run in order 10, 6, 2, 4, 8).
    (d) Shortest job first.

    For (a), assume that the system is multi-programmed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

17. In a round-robin scheduler, how does the choice of time quantum interact with the cost of context switching, and how does this relationship impact performance? (4 pts)

18. A real-time system has four periodic events with periods of 50, 100, 200, and 250 msec each. Suppose that the four events require 35, 20, 10, and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable for rate-monotonic and earliest deadline first scheduling. (3 pts)