

Homework 1 - System Programming

Instructor for CS 241: Klara Nahrstedt
Deadline, March 6, 4pm
Department of Computer Science
University of Illinois, Urbana Champaign

The homework 1 is an **individual effort activity!!** (no pairs). Each student submits his/her own solution in the electronic form using the PDF format. The format should use 11pt or 12pt font. Each student submits his/her work to Illinois Compass in the same manner as he/she submitted machine problems:

1. Log into Illinois Compass and select cs241 from the *Course List*.
2. Select *HW1 Handin* from the CS 241 Home Page
3. Click the *Add Attachments* button in the *Submissions* section.
4. In the *Choose Files* window, click *My Computer*.
5. Select your PDF document in the file selection dialog.
6. Click *Submit*, then check *OK* in the confirmation alert.
7. On the *Confirmation* page, click *Continue*.

1 Problems on Process/Thread Management (20 Points)

2

- a. (3 Points) If a system has only two processes, does it make sense to use a barrier to synchronize them? Why or why not?
- b. (2 Points) In a system with threads, is there normally one stack per thread or one stack per process? Explain.
- c. (2 Points) What is a race condition?
- d. (9 Points) What are the three main states that a process can be in? Describe the meaning of each one briefly as well as the relation among them (i.e., explain the transition conditions among these three states).
- e. (4 Points) Can two threads in the same process synchronize using a kernel semaphore if the threads are implemented by the kernel? What if they are implemented in user space? Assume that no threads in any other processes have access to the semaphore. Discuss your answer.

2 Problem on Synchronization (20 Points)

Show how counting semaphores (i.e., semaphores that can hold an arbitrary value) can be implemented using only binary semaphores and ordinary machine instructions (you may use pseudo-code for the implementation). Explain the role of each semaphore that you use in the implementation of the counting semaphore.

3 Problem on Synchronization (10 Points)

- a. (6 Points) Give a sketch of how an operating system that can disable interrupts could implement semaphores. (You should not use the semaphore.h file, etc)
- b. (4 Points) Does the busy waiting synchronization solution using the *turn* variable (see strict alternation solution, Figure 2-10 in Tanenbaum Book) work when the two processes are running on a shared-memory multiprocessor, that is, two CPUs, sharing a common memory?

4 Problem on Scheduling (30 Points)

Consider the workload in Table 1:

Process	Burst Time	Priority	Arrival Time
P_1	50 ms	4	0 ms
P_2	20 ms	1	20 ms
P_3	100 ms	3	40 ms
P_4	40 ms	2	60 ms

Table 1: *Process Workload*

- (18 Points) Provide schedule using *preemptive Shortest Job First*, *non-preemptive Priority*, (a smaller priority number implies higher priority) and *Round Robin* with quantum 30 ms.
- (12 Points) What is the average waiting time of the above specified scheduling policies?

5 Problems on Scheduling (20 Points)

- (8 Points) A soft real-time system, running Earliest Deadline Scheduling policy, has four periodic processes with periods of 50, 100, 200, and 250 msec each. Suppose that the four processes require 35, 20, 10, and x msec of CPU time, respectively. What is the largest value of x for which the system is schedulable?
- (12 Points) Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and X . In what order should they be run to minimize average response time? (your answer will depend on X).¹

¹Note the difference between the waiting time and the response time. The *process response time* is measured as the time interval from the time point when a process enters the system (e.g., command is issued and process enters the ready queue) until the time point when the process yields a desired result. The *process waiting time* can be one or a sum of process waiting time intervals where a waiting time interval is measured from the time point when a process enters the system (enters the ready queue) until the time point when it is scheduled and grabs the processor again.