ECS150
WQ2004
Homework Assignment 2
Due Tuesday, Feb. 17, 2004, 10AM, HW Box, This due date and time is firm

Note: The assignment is rather long, as it covers the following areas: system calls, scheduling, Minix kernel, deadlock, and low level inter-process communication; the latter topic I will cover in class on Tuesday. I have attempted to indicate the topic covered by each question, so you can start on it immediately. Please, do wait until the night before the due date to start on the problems. We will attempt to post solutions just after the 3-day President's Day weekend, giving you a bit of time to view the solutions as you study for the MT. I will tell you more about the MT next week, but for now assume the following topics will be covered: multiprogramming concepts, system calls and programs that call them, scheduling, Minix kernel, deadlock, low-level inter-process communication (which I will cover this coming Tuesday).

1. Tanenbaum Chapter 2, Number 20 -- scheduling

2. Tanenbaum Chapter 2, Number 21 – scheduling
3. Tanenbaum Chapter 2, Number 23 – scheduling

4. Here is a question about scheduling. Real-time systems are characterized as systems that have certain processes that must complete execution prior to some time deadline. The following table gives the service time and deadline for each of 5 processes.

process	Service time	deadline
A	350	575
B	125	550
C	475	1050
D	250	none
E	75	200

I. Assume the scheduling policy attempts to meet all deadlines, which process is a good candidate to be scheduled last? Explain.

II. Show on a timing chart, two possible schedules such that all deadlines are met.

III. Briefly describe an algorithm that determines if given the service and deadline times for N processes, can all deadlines be satisfied.

5. Here is a question on scheduling. It refers to the definitions I introduced to compare scheduling algorithms, which also appear in Olsson's notes; I summarize the definitions on at the end of the question.

a) Consider the FCFS policy. Assume there are n processes in the system when a new process requiring t seconds of service time arrives. The average execution time for a process is τseconds. Determine the following for the newly arriving process; again, it requires t seconds of service time.

I. Time the new process waits until it first receives service
II. Response time
III. Missed time

b) Now consider the RR policy, where the quantum is q. Assume there are always n processes in the system (where a process on the system is either on the ready queue or on the cpu). Again, let us define an average process as one that requires τseconds of service time. Determine the following for a newly arriving process requiring t seconds of service time:

i. Time the new process waits until it first receives service
ii. Response time
iii. Missed time

c) Discuss how FCFS compares with RR by indicating a relationship between q and τ such that the RR is superior with respect to *first response time.*

d) Now consider a modification to round robin scheduling, which we call MRR. In MRR double the time quantum given to a (long) process each time it passes through the ready queue. Again, assume n short processes (each of which requires a service time of exactly q seconds). Assume the long process requires a service time of L seconds. Assume a situation where the long process is in its final turn at the cpu. Determine the missed time and penalty ratio for the long process and a short process that arrives just as the long process is getting its final turn at the cpu.

Here are the terms about scheduling you will need for this problem:

A process requires t seconds of service time.
"Time to first response" is the first time the process sees the cpu
"Response time", T, is finish time - arrival time
"Missed time" is T-t.

6. Here is a question on low-level process synchronization. Suppose two processes, P1 and P2 are multiprogrammed, where the code associated with each process is as follows:

```
     P1: a:= a + 1;
      b:= b + 1;
  P2: b := 2*b;
      a:=  2*a;
```

Suppose the two items of data, a and b, are to be maintained in the relationship a = b. Determine an interleaving of the instructions for P1 and P2 such that this relationship is not maintained. Assume, for this problem, that each of the four high-level instructions are atomic.

7. Tanenbaum Chapter 2, Number 29 - Minix kernel

8. Tanenbaum Chapter 2, Number 31 - Minix kernel

9. Tanenbaum Chapter 2, Number 32 - Minix kernel

10. Here is a question on the Minix kernel. It involves the actions of the kernel in processing a system call.

a. Assume a task is running. Is it possible for the task to make a system call? Explain.

b. Assume a server is running. Is it possible for the server to make a system call? Explain.

c. What kernel function is called first in processing a system call? Is this function written in assembly or C? Why?

d. Assume a user process P1 is running and makes a system call. As a result of the system call, a message is sent. What is the soruce and destination of the message? On what line in the Minix source code is the call to send?

e. Revisiting (d), is it possible for the destination to be busy when the message is sent? Explain.

f. What process is likely to be scheduled immediately following the system call? Explain.

11. This is a question about the Minix kernel.

a) Assume the clock task is running. Is it possible for a clock interrupt to come along at this time? What about a keyboard interrupt? Explain.
b) Now let us consider what happens in the kernel while handling this interrupt.

I. Will the running clock task be preempted as a result of the interrupt? Explain.

II. Will the task associated with the interrupt be placed on the ready queue while the current task is still on it? Explain.

III. Explain how the occurrence of the interrupt is recorded by referring to specific lines of code in the Minix kernel.

IV. If the answer to b-II is *no*, when is the task associated with the interrupt placed on the ready queue? Explain with reference to specific lines of code in the kernel.

12. This is the question about the Minix kernel. Assume 3 processes (user processes, tasks or servers), which we can refer to as A, B, and C call *mini_send*, in order and all desiring to send a message to process D which is busy and has not called *mini_rec.* Thus, each of the processes A, B, C will block.

a) Show the relevant values for the proc tables for A, B, C, and D after the 3rd process has called *mini_send.*
c) Now assume D has called *mini_rec(D, ANY, mD).* Show the relevant proc tables after this call.
d) Now assume D has called *mini_rec(D, C, mD).* Show the relevant proc tables after this call.
e) In general, is it possible for the queue of processes blocked on sending to a process that has yet to do a receive overflow? Be very specific on how large the queue can become, and where the queue is implemented in Minix.

13: Tanenbaum, Chapter 3, Number 11 -- deadlock

14. Tanenbaum, Chapter 3, Number 13 -- deadlock

15. Tanenbaum, Chapter 3, Number 14 -- deadlock

16. Tanenbaum, Chapter 3, Number 15 -- deadlock

17. Tanenbaum, Chapter 3, Number 18 - deadlock

18. Here is a question about deadlock. Consider the following resource graph.

Here are some definitions that I presented in class:

A process is blocked in a given state if it can neither request, acquire or release resources in that state.

A process is deadlocked in a given state S if it is blocked in S and no matter what operations by any process(state changes) occur in the future, the process remains blocked.

S is called a deadlocked state if there exists a process p deadlocked in S.

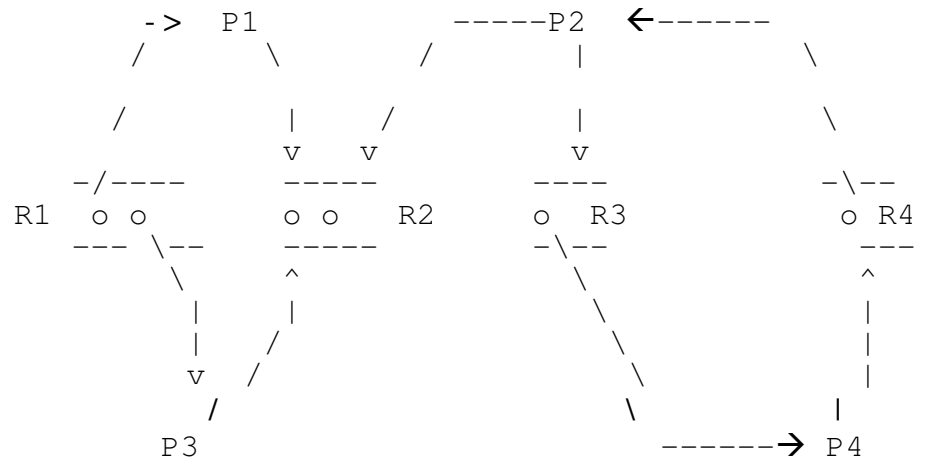
Now here are some questions about the above resource graph:

a. Which processes (if any) are blocked? Explain.

b. Which processes (if any) are deadlocked? Explain.

c. Is the state a deadlock state? Explain.

d. Which additional processes (if any) might be deadlocked in some future state? Indicate the operations that would make this happen.



19. This question is about deadlock. Recall the *dining philosopher* problem, where 5 philosophers are seated around a round table with a fork between each philosopher and its neighbor. When a philosopher desires to eat he makes the following requests:

request fork to my left
request fork to my right
eat
release fork to my left
release fork to my right

Requests are honored if the resource is available, or the request is denied and the requesting process (philosopher) has to block. Now consider some variants of this problem

i. Imagine that in addition to the fork resources, there is a room that allows no more than four people inside. Now the actions of a philosopher are:

request room slot
request fork to my left
request fork to my right
eat
release fork to my left
release fork to my right
release room slot

Show the resource graph after five processes have requested a room slot and all four that
were successful tried to grab their left fork.

ii. Is deadlock possible for the situation described in (a)? If yes, give an example of requests that lead to deadlock or, if no, indicate which of the four conditions is not satisfied.

iii. For this part, assume there is no room resource but a bowl of spaghetti in the middle of the table, which is accessible by only one philosopher at a time. The actions of a philosopher are:

```
request fork to my left
request fork to my right
request bowl of spaghetti
eat
release fork to my left
release fork to my right
release bowl of spaghetti
```

Is deadlock possible for this situation? If yes, give an example of requests that lead of deadlock or, if no, indicate which of the four conditions for deadlock is not satisfied. Is there concurrency in this case? Explain.

20. This question concerns low level inter-process communication. Consider a concurrent program with two processes, P and Q, shown in the following code. A, B, C, D, and E are arbitrary atomic (indivisible) statements. Assume that the main program (not shown) does a cobegin of the two processes.

```
Procedure P;
Begin
 A;
 B;
 C;
End


Procedure Q;
Begin
 D;
 E;
End
```

Show all possible interleavings of the execution of the preceding two processes; show this by giving execution “traces” in terms of atomic statements.

21. This question concerns low-level inter-process communication. Consider the following program, the purpose of which is to fetch a

character at a time from the keyboard, buffer it, and then output it to a display. Someone had the brilliant idea that the three operations could be performed concurrently. Here is the program.

```
Procedure echo;
 var in, out: character;
   begin
     input(in, keyboard);
     do true ->
       cobegin
          out:= in;
          output(out, display);
             input(in, keyboard)

       coend
    od
  end
```

Successive calls to input return, in sequence, the characters c1, c2, c3, … Of course, we would like the exact sequence of characters be sent (via output to the display. Suppose that the first character has been successfully read in and displayed. Thus, the input sequence is c1, as is the output sequence; this means that the three processes associated with the cobegin statement are executed in the order given – for the first time through the loop. Assume that the three statements inside the loop can be interleaved, but not overlapped.

B. What are the possible outcomes of the second iteration of the loop; be sure to show all possible interleavings and for each interleaving the value of the character output to the display. Is the output for all interleavings as desired?

B. Give a very easy fix to the above program so that the correct value is output to the display for all iterations; your solution should still allow for some concurrent activity.

22. This question concerns system calls. Describe a simple program that determines the length of a pipe. The core idea is very simple, as you will discover. However, I would like your program to print out the desired number and then gracefully terminate itself; that is it should not be necessary for a user running this program to send it a signal to kill its process.