MIDTERM with SOLUTION
March 3, 2005

Comp.Sys.Org.II, V22.0202.003, Yap, Spring'05

1. PLEASE READ INSTRUCTIONS CAREFULLY.
2. Answer all questions in the space provided.
3. Use the REVERSE SIDE of exam paper for scratch work.
4. This is a closed book exam, but you may refer to one $8'' \times 11''$ sheets of prepared notes.

GENERAL COMMENTS ABOUT YOUR ANSWERS:

- *Question 1 asked for "brief justifications" but many simply wrote down an answer and left it at that! Question 2 asked for intermediate trees after EACH insertion/deletion/rotation, but some simply skip many crucial intermediate steps. Please realize that you are not entitled to any points when you ignore such explicit instructions (but we have graded more generously than this).*

QUESTION 1. SHORT QUESTIONS. (5 Points each)

1. What is the difference between traps and interrupts?

   ANSWER: Traps are the usual way for users to make system calls (to get into kernel mode). Interrupts are for the system to stop ("interrupt") a process so that it can carry out some task instantly (e.g., in I/O handling).

   Of course, there are also many similarities between the two, in their use of vectors and causing the saving of states, etc.

2. Consider the following lines in a file called "Makefile":

```
mytest:  wonderful.h amazing.c                #line 1
-------> gcc -o amazing amazing.c            #line 2
                                              #line 3
yourtest: great.c stunning.c                  #line 4
-------> gcc -c -o great.o great.c           #line 5
-------> gcc -o stunned great.o stunning.c    #line 6
-------> stunned                              #line 7
```

   where `------->` indicates a tab character. In the terminology of the make program, what is the term that describes `mytest` and `yourtest`? What term describes `wonderful.h` and `amazing.c`?

   ANSWER: They are called "targets" and "dependencies", respectively.

3. What happens when you type "`make yourtest`" in the directory containing the above Makefile? Please use the terms "assemble, link, load" in your answer.

   ANSWER: It will assemble but not link `great.c` and put the result into a file `great.o`. Next, it will assemble `stunning.c` and also link the result with `great.o` and put the executable code into `stunned`. Finally, it will load `stunned` and execute it.

4. List some properties of the Petersen Solution to the Mutex problem that might be considered undesirable?

   ANSWER: Petersen uses busy-waiting, requires shared variables, and each process knowing its own unique process ID.

   There is also the possibility of priority inversion, and even lock out, in case processes have priorities.

5. A real-time system has four periodic events with periods of 50, 100, 200, 250 msec each. Suppose that the four events require 35, 20, 10 and x msec of CPU time. What is the largest value of x for which the system is schedulable?

   ANSWER: We need $\frac{35}{50} + \frac{20}{100} + \frac{10}{200} + \frac{x}{250} \leq 1$. Thus the largest possible x satisfies $0.7 + 0.2 + 0.05 + (x/250) = 1$. So $x = 12.5$.

6. A system with 4 processes and 5 allocatable resources are shown:

	Allocated	Maximum
Process A	1 0 2 1 1	1 1 2 1 3
Process B	2 0 1 1 0	2 2 2 1 0
Process C	1 1 0 1 1	2 1 3 1 1
Process D	1 1 1 1 0	1 1 2 2 1

   Suppose the currently available vector is (0 0 x 1 1). What is the smallest value of x for which this is safe? ALSO, give the order in which these can finish with this value of x.

   ANSWER: The smallest value is $x = 2$. The order of finishing is DCAB or DCBA.

   FIRST, let us show that the problem is not safe when $x = 1$: only D can finish. The available vector is now (1 1 2 2 1). But then no other process can finish.

   NEXT, if $x = 2$, then we can can again let D finish first. Then the available vector is (1 1 3 2 1). Now Process C can finish, and the available vector becomes (2 2 3 3 2). Now both A or B can finish (in either order).

   NOTE: This problem was from the text book (one of our study problems). But there was a slight error in the text which we have fixed by changing a single number.

7. Suppose the page reference string is 0172, 3271, 03 (I put commas to make the long string easier to visualize). How many page faults do you get if there are 4 page frames and you use LRU page replacement policy?

   ANSWER: 7 page faults. 5 faults to when we reference the pages 0,1,7,2,3. The next references for pages 2, 7, 1 causes no page faults. Finally, we get two more page faults for pages 0 and 3.

QUESTION 2. SOME SMART IDEAS. (45 Points)

1. Student Smart says "If you want to schedule processes to minimize the TOTAL wait time of all processes, it is quite easy. Just use the Shortest Jobs First rule." *Show that Smart is right. But say why this is not the end of story.*

   ANSWER: Smart is right: here is the proof. Suppose P takes time T and P' takes time T' where $T > T'$. Let us compare the total wait time S_1 when P is scheduled ahead of P', with the total wait time S_2 when P' is scheduled ahead of P. We see that $S_1 - S_2 = (2T + T') - (2T' + T) = T - T' > 0$. Thus S_1 is more than S_2.

   There are two problems:

   (1) This solution assumes that WE KNOW THE RUNNING TIME OF EACH JOB. But we don't usually. To get this idea to work, we must have some ways to estimate the "expected" run time of each job. For instance, we can use past executions of each program (perhaps with "aging") to estimate the expected run time.

   (2) If the jobs arrive at different times, we need to use preemption to minimize the total wait time. I.e., we now look at the remaining time to completion of job, and schedule the one with the least remaining time.

   This raises an issue of fairness: what if a long job is REPEATELY preempted? Say a 2-unit job is repeated preempted by the arrivals of many 1-unit jobs. Now it is not so clear that total wait time is the best policy. Of course, policy issue is not easily decided.

2. Again, Smart says: "Mutual exclusion is really a simple to achieve. Say we have n processes whose ID's are $0, 1, \ldots, n-1$. We have an integer variable (TURN) which each process checks before entering a critical section (CS). If TURN is equal to my process ID, then I go into the CS. When I am done, I increment the TURN variable (modulo n)." *Show that is Smart right, but say why we are still not happy.*

   ANSWER: Smart is right – his solution DOES achieve mutual exclusion. Only the process whose ID equals the TURN variable can be in the CS.

   The problem is that it fails in other aspects – here, I am refering to the 4 requirements for an acceptable mutex solution (see p.102 of textbook). YOU MUST REFER TO THESE REQUIREMENTS FOR FULL CREDIT.

   E.g., one requirement is that a process not in the CS should NOT prevent others from entering the CS. But if the process whose ID equals TURN remains outside the CS, he will prevent all others from entering CS.

   NOTE: Peterson's solution gets about this unfairness problem by augmenting the TURN variable with another variable with the INTERESTED flag.

3. Another Smart quote: "Deadlock prevention is a no-brainer. Just make sure that there is a global ordering on every resource. Processes must request each resource according to this ordering. If any request fails, the process must release all its acquired resources and begin again." *Show that Smart is correct, but discuss why we may not like this solution.*

   ANSWER: Why is Smart correct?

   Many of you pointed out correctly that there is no deadlock because one of the 4 CONDITIONS for deadlocks is missing (i.e., not giving up a resource once you acquired it). See p. 164 of Textbook. YOU MUST REFER TO THESE REQUIREMENTS FOR FULL CREDIT.

   But we need to be careful – there can be **general starvation** where each process repeatedly gets one resource but releases it again because it cannot get the next one. [Give a scenario where this actually happens]

   This DOES NOT OCCUR in our example because of another property in our solution (which no one seem to have noticed): we have a global ordering of the resources. This means that there is always one process that holds the largest acquired resource. This process will never wait on any one else! So there is no starvation either! YOU MUST REFER TO THIS PROPERTY TO GET FULL CREDIT.

   Why might this be an undesirable solution? There are two problems.

   (1) Inefficiency: it may not be a good idea to release all acquired resources just because you cannot get another one – there may be no deadlock ahead! In practice, it seems to accept the SMALL possibility of deadlock by not releasing acquired resources.

   (2) We may have **individual starvation** where a process that needs MANY resources may be starved by other processes requiring few resources. (Thus, even though general starvation is not possible, individual starvation can exist).