

COSC 2P13 Assignment 2 Spring 2011

Due: July 4, 2011 at 12:00 noon.

Late date: July 7, 2011 at 12:00 noon, subject to a 25% penalty. Submissions later than this will receive a grade of 0.

Values of questions shown in square brackets, totaling 90. This assignment is worth 15% of your final grade.

1. [5] A computer system has enough room to hold six processes in its main memory at once. Each of these processes is IO bound and spends 75% of its time waiting for I/O. What fraction of CPU time is wasted?
2. [6] A system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:

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

If the available resources are $(x\ 0\ 1\ 2\ 2)$, what is the smallest value of x for which this is a safe state? Demonstrate how you arrived at your answer.

3. (a) [3] A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.

(b) [3] Consider the problem in (a) again, but now with p processes, each needing a maximum of m resources and a total of r resources available. What condition must hold to make the system deadlock free?
4. [6] Swapping systems eliminate holes by compaction. Assuming a random distribution of many holes and many data segments, and the time to read or write a 32-bit memory word of 5nsec, about how long would it take to compact 6GB? For simplicity, assume that word 0 is part of a hole and that the highest word in memory contains valid data. What about if you upgrade to memory that can be read or written to in 2nsec? Comment briefly on the effects of compaction on normal processing.
5. [6] The number of instructions executed between page faults is directly proportional to the number of page frames allocated to a program. If the available memory is doubled, the mean interval between page faults is also doubled. Suppose that a normal instruction takes 3 μ sec, but if a page fault occurs, it takes 2003 μ sec (i.e., 3 μ sec for the instruction, plus 2000 μ sec just to handle the fault). If a program takes 60 seconds to run, during which time it gets 8000 page faults, how long would it take to run if twice as much memory were available?

6. [10] Compare the storage needed to keep track of free memory using a bitmap versus a linked list. The 4 GB (4096 MB) memory is allocated into units of size n bytes. For the linked list, assume that memory consists of an alternating sequence of segments and holes, each 64KB long. Also, assume each node in the linked lists requires a 32-bit memory address, a 16-bit length field and a 32-bit next-node field (10 bytes in size total). How many bytes of storage are required for each method? Which is better, in which cases? Discuss the implications of the size of the allocation unit, i.e., what is the effect of n and give the “cross-over point” (the value of n where one solution becomes better than the other).
7. a) [3] If an instruction takes 4 nsec and a page fault takes an additional n nsec, give a formula for the effective instruction time, assuming a page fault occurs every k instructions.
 b) [3] Now, assuming $n = 20$ nsec, demonstrate how page fault frequency affects system performance by using your formula to compare $k = 3$, $k = 15$ and $k = 40$.

Part B – Programming

8. [45 total] A shell is a command line interpreter, a basic user interface to a computer operating system. For this assignment, you will develop the beginnings of a simple shell by implementing the basic functionality outlined below. Additional functionality will be added in Assignment 3 (so don't delete your code after submitting Assignment 2!!!).

The shell should continuously prompt the user for input, which is then read via *scanf* (or similar input function). The prompt should be your initials followed by “sh-\$”, e.g., Joe Blow’s shell would prompt “jbsh-\$ ”. The shell should initially support the following commands:

- a) `cd <directory>` - change current directory to <directory>. If the <directory> argument is not present, it should print the current directory.
- b) `clr` - clear the screen
- c) `dir <directory>` - list the contents of <directory> or list the contents of the current directory if <directory> is not provided. This should provide details such as file permissions, modify dates, file sizes, etc.
- d) `pause` - pause the shell until a key is pressed
- e) `quit` - quit the shell
- f) General purpose functionality - your shell should support other basic Linux commands either via full path specification or by using the system \$PATH variable. For example, the `ls` command is located in `/bin/ls`, hence this command could be run in your shell as `/bin/ls`, followed by arguments such as `-l`, `-a` etc.

All command should use *fork* and *exec* as appropriate to execute the appropriate Linux command. Command arguments should be parsed as strings and used as input to the *exec* system call. You may assume that arguments will be separated by spaces. For example, `ls -l -a` produces the same results as `ls -la`, but you only need to support the

former). Other functionality can be implemented using standard C procedures (available via specialized libraries). You will have to do some research for these functions.

Tips:

- 1) Start early – there will be no extensions.
- 2) Look at the C string manipulation functions, you will need to read in the strings the user types, tokenize the arguments, and store them in an array of strings. You may assume that no line of input will ever be greater than 50 characters (i.e., use a char array, rather than a char* – this should be a bit easier this way).
- 3) Examine the Linux man pages for the functionality of various similar commands – effectively, you are aliasing some of these with your own command names

Assignment Submission Requirements

All assignments must be submitted BOTH in hardcopy (print out of any source code and a few samples of its execution) AND in softcopy (electronically) using the command **submit2p13** on Sandcastle. This program will copy the contents of the current directory to the marker's account.

All hardcopy submissions must satisfy the following:

- The submission must be **typed and printed** on a laser/inkjet printer
- All the papers must be stapled together
- A standard assignment cover-page (<http://www.cosc.brocku.ca/forms/cover>) should be printed, filled out completely, signed and stapled to the front of the outside of the assignment (optionally in an 9x12 envelope).

Submit your assignment to the Assignment Box outside of J332, in COSC 2P13 slot, before the due time indicated above. Only one submission should be made per assignment.

- Unless otherwise specified, all assignments are due at 12:00 PM on the due date.
- Both the hardcopy and the softcopy submissions must be done before the deadline.
- Differences between the hardcopy and the softcopy submissions will be penalized.
- Familiarize yourself with the department's policy on plagiarism and the university regulations on plagiarism and academic misconduct. In particular, all referenced sources (for code, written material, etc.) **must** be cited.

Assignments that do not follow these requirements will NOT be marked!