CS3102 Operating Systems
Laboratory 1

1. Objective.

The primary objective of this laboratory exercise is to provide hands-on experience in using a UNIX system so that students can get themselves familiarized and prepared for subsequent laboratory work.

2. Environment.

The development environment for all lab works is SUN workstations running operating system Solaris which is a version of UNIX System V Release 4 and act as hosts. Students will use PCs running terminal emulation (e.g. Telnet) to login as terminals. C is the language used for all the lab work throughout this semester. It is also possible for students to use dialup lines to login to the system remotely provided appropriate terminal emulation software is used.

3. Activities.

a) Boot up your PC using account osuser”, password sedos” and Win95 will be started. From start button, select run and enter elnet”, type in slab” for remote system when prompted. This will connect you to one of the Unix machines in CS Lab running OS Solaris. You should already have your own login id in CS lab.

b) The next important thing is to know how to find reference information for commands. The online reference manual can be invoked by command man. Want to know how to use the man command? Try man man.

c) Try out the following commands and record the result :-

- ls, ls -l, ls -al, ls -al/etc
- who, who -H, who am I
- date, date +%d, DATE
- cp
- cd
- pwd

d) Try out the mail command and send mails to your classmate, where can you get help information for the mail command? How to do automatic mail forwarding?

e) Key in the following input ([ ] stands for control keys) :-

- cat
  aadlfkldfkldfkjdf
  I am typing something
  this line 3
  [ctrl-d]
  Can you find where these lines are stored?

- cat > myfile
  aadlfkldfkldfkjdf
  I am typing something
  this line 3
  [ctrl-d]
  ls
  cat myfile

f) Try using the vi editor to create and edit text files (refer to the attached summary or look at the online manual). vi editor is quite user-unfriendly, if you are not happy with the vi editor, you may try out other editors such as PICOS. Another alternative is to use a PC text editor (e.g. notepad or edit etc.); you have to do this by logout (exit or Cntl-d). After editing, you have to transfer your text file from your local PC to the UNIX host (use ftp option from the main menu), you should familiarize yourself with this procedure if you decide not to use vi editor in your future work. There are also other editors for unix available.

4. Create the following little C program, compile and link it using the cc command. Find out the detail syntax of cc command by yourself. Try execution and see if you get the correct result.

#include <stdio.h>

main()
{
  int x, y, z;
  printf("Enter first integer\n");
  scanf("%d", &x);
  printf("Enter second integer\n");
  scanf("%d", &y);
  z = x * y;
  printf("Sum is %d\n", z);
}

This is your first C program. If you have not yet learned C, try to guess the meaning of the syntax.
CS3102 Operating Systems
Laboratory I
Quiz

Hand in this sheet to your lab tutor at the end of the lab sessions.

1. Some commands have the same name as a system call e.g. times, how can you get help information for each of them?

2. A Unix command can be regarded as 3 components: command, option and argument. Use an example you have learnt from this lab to illustrate this.

3. Give the command line that can display the current system date and time in the following formats:
   - Date as mm/dd/yy
   - Time as hh:mm:ss
   - Gregorian Date

4. Unix is a?
   - Multitasking system
   - Multiuser system
   - Timesharing system

5. What are the default input and output file names for compilation and linking?

6. Do you know what are the phases involved when using the cc command? Try recompile your program by using option -d.

CS3102 Operating Systems
Laboratory II

1. Objective.
   To learn and have hands-on exercise with Unix SHELL.

2. Background.
   Shell is an interface that provide interaction and system services to users. As Unix developed, there were different interfaces and three most famous ones were: Bourne, C and Korn Shells. Bourne shell was developed by Stephen Bourne in the early 1970s. This is the earliest shell and has been available in all versions of Unix. It has the smallest implementation of the available shells and has always been improved upon by AT&T Bell Laboratories.

   C Shell was developed by Bill Joy in the late 1970s at Berkeley. This shell was an improvement over the Bourne shell as it had additional features such as command history and a syntax very similar to C programming. Commands and syntax are very much different and not compatible. To keep up with the competition, David Korn at AT&T Bell Labs based on the original Bourne shell, developed the Korn shell incorporating many of the user friendly features similar to C shell. It can also support most of the Bourne shell commands which makes transition easier.

   By design, the nature of Shell is to provide an interface which can be changed by different users. Each user can choose a shell which suit him/her best.

3. Activities.
   a) Login the system, note the prompt character, type command ps, this will show all processes belong to your session, do you know what shell are you running?
   b) Key in the following :-
      
      sh
      csh
      ksh
      ps

      Note the change of prompt characters, what information can you get from the ps command? How many shells are you running? Key in :-

      exit
      exit
      exit
      exit

      Can you explain what is happening?
   c) Key in the following :-
can be grouped together to become programs called scripts. Very sophisticated systems have been developed by using only shell scripts.

f) Key in :-

```
cat > letter
this is a mail send to myself
crtl-d
```

cat > mailist
... your own login id...
... your own login id...
... your own login id...
crtl-d

```
mail 'cat mailist' < letter
echo "the names in list are : $(cat mailist)"
echo "the names in list are : $(< maillist)"
```

Use the mail command to see if you get 3 mails. Figure out what is happening. This shows the concept of command substitution in shell and it is a very flexible and powerful features. (Ask lab tutor to explain if you have difficulty in understanding the concept, note that the quotes for command cat mailist is backquote not a single quote)

g) Writing shell scripts.

Key in :-

```
cat > myscript
echo 'please enter the input filename followed by the output filename' read file1 file2
echo Input: $file1
echo Output: $file2
crtl-d
```

```
myscript You should get an error
is -l myscript
chmod +x myscript
```

h) Read the given reference and try out the given example scripts which show parameter substitution and if-then-else statements.

i) Write a script that takes a filename given as an argument, determines the “file type” of the file, and if it is a commands text file, an ASCII text file or an English text file, display the file page by page. If the file is any other type, display the message “file filename is not displayable”. If the file does not exist, display “file filename does not exist”.

Submit your script for (i), answers for (b), (d), (c) and (f) at your week 5 lab sessions.
1. Objective.

To understand the concept of process management and get familiar with process related commands. To develop a more complex C program so as to "warm-up" your programming skill.

2. Activities.

All activities for this lab will be run under Korn Shell. Process structure in Unix is like its file system, arranging in a hierarchical fashion. It has parents and children, even a root. A parent process creates a child process, which can also create other processes. The first process started when the system is booted is init, which is the ancestor of all processes.

a) From online manual, study various options of the ps command, try out the following and note the result

   `ps`
   `ps -f`
   `ps -l`
   `ps -a`
   `ps -u [some user name]`

   Note the meaning of various fields.

b) Execution of multiple commands

   `date ; pwd ; who`

c) Grouping of commands

   `(pwd, ls -al) > file.1st`

   `pwd`
   `(cd /etc ; ls -al)`
   `pwd`
   `cd /etc ; ps ; ls -al ; ps`

   Can you explain the difference between multiple commands execution and grouping of commands?

d) Execute commands in background

   Usually, the shell remains inactive during command execution. Another command cannot be executed until the previous command has completed and the shell displays another command prompt. This is referred to as foreground execution. Alternatively, commands can be executed in background. While a program runs in background, the shell is immediately available to execute another command.

   `ls -al /etc`
   `ps`
   `pwd`
   `ls -al /etc&`
   `ps`
   `pwd`

   When do you think background execution is most useful?

   `sleep 600`
   `Ctrl-z`
   `bg`

   `jobs`
   `ps`
   `fg`
   `Ctrl-z`
   `bg`

e) Terminate processes

   Study from online manual, the kill command.

   `ls -al /etc > file.1st&`
   `ls -al /etc > file.2nd&`
   `ls -al /etc > file.3rd&`

   How can you terminate the second process?

f) Process scheduling

   `at now +2 minutes`
   `date`
   `echo here am I !`
   `Ctrl-d`
   `at -l`

   After the process come out, do the following :-

   `at now +2 minutes`
date
echo here am I!
Ctd-d
at now +2 minutes
date
echo here am I!
Ctd-d
at -1

How can you remove the job from the backlog? Propose a way to schedule a job to run repeatedly every 5 minutes.

g) A C program to study the effect of fork().

You are required to code a program CREATOR that will create, kill and examine processes. The program accepts 4 commands and their specifications are as follows:

i) child [option] start-time life
   A child process is created after start-time seconds from now and the child process will run for life seconds. Option is optional and if exists, can only take the value -b. When -b option is specified, the child process is run in background and CREATOR will ask for other commands without waiting for this command to finish. If -b is not specified, CREATOR will wait until the command is finished before another command is accepted.

ii) grandchild [option] child-start-time child-life grandchild-start-time grandchild-life
   Same as in (i), a child process is created after start-time and runs for life. A child (grand child) will also be created by the child process after grandchild-start-time relative to the start of the child process and runs for grandchild-life seconds. The option parameter is the same as in (i). If -b is not specified for option, CREATOR will wait for the child to finish before accepting a new command.

iii) kill [option] pid
   The child process with id equals to pid will be terminated. Option is the same as in (i).

iv) status [option]
   Print out information for all the running child processes. Information displayed should be similar to that output by the ps command. You may use the ps command or program your own coding to do this.

v) end
   Terminate CREATOR and all child/grandchild processes that are running.

Whenever a process starts, process id and start date-time should be displayed. Upon termination, the process id and cpu usage time in seconds should be displayed. You should make it possible to display all these information either on screen or to a file.

By adjusting the parameters, the effect of different situations can be studied e.g. parent terminating before child.

Your program should validate input commands and should not terminate abnormally. Useful system/library calls are: fork, exit, wait, sleep, system, times, printf, scanf. You should check out, by the man command, the definitions of these system calls and which header files should be included. Any standard C library functions can be obtained from any C programming reference book.

At the end of lab session, submit answers for all short questions asked in the activities. The e program will be due on week 8.
CS3102 Operating Systems

Laboratory IV

1. Objective.
- To introduce the OSP package, a simulator for an OS
- To investigate how interrupt handling are performed in a typical OS
- To write a simple interrupt vectoring routine
- To write 3 simple ISRs to handle OSP's process management system calls

2. Introductory Notes.
2.1. Although sufficient handouts will be provided, you are encouraged to obtain a copy of the OSP booklet so that you may have a better overview of all the features in OSP. The architecture of OSP, although is highly simplified compared to a commercial OS, can help you to understand concepts and algorithms presented in the lectures.

2.2. Features of OSP.
The whole OSP is actually a large C program which simulates the behavior of a simplified OS running in a hypothetical computer. You may view OSP as 3 main parts:

a) the simulated operation environment representing the real world activities such as user programs making system calls, occurrence of different kinds of interrupts. These activities are non-deterministic. The module to generate these activities is SIMCORE, by using a random number generator, it produces random (nearly) events by simulation. In addition, SIMCORE will also keep track of all the events it generated and record down the expected correct result which will then be compared to the results produced by the other OS modules (i.e. part (c) below, some modules will be written by you). Any warning and error will be shown.

b) DIALOG.C - module to assist debugging, it prints out "snapshots" about the internal system tables e.g. ready list so that you can trace whether the modules you have developed are correct or not.

c) the OS modules that together form an OS running in a hypothetical computer. Each module can be regarded as an OS component mentioned in our lectures.

2.3. Functioning of OSP
As OSP is just a C program running in UNIX, all occurrence of interrupts are simulated and implemented as function calls. There will be no actual context switch. You may view SIMCORE is the main program and according to a simulated event list, calls different OS modules which are implemented as C functions.

2.4. Some OSP terms
There are 2 terms you have to clearly distinguish: external and internal routines. When you develop an OS module, you are actually writing up some C functions (routines) which will be called by other modules. These routines with respect to your OS module are called internal routines. You also have to call other routines in other modules (provided by OSP already) in order to complete your work, the routines you called are external routines.

3. Activities.
In this laboratory session, you will implement the interrupt vectoring module and routines to handle 3 process management system calls: process start, process kill and process termination. You will write codes to satisfy the descriptions given in the OSP Programmer's Manual making use of what you have learned about operating systems to fill in the missing details. The module you write will be linked with the rest of OSP, and tested with various simulation parameters. This lab exercise looks complicated but in fact the codes you are going to write are quite trivial. One of the objectives of this exercise is therefore to let you get familiar with OSP, to know the control flow within OSP and prepare for the next lab.

Before attempting to do this laboratory exercise, make sure that you have studied the handouts about OSP in detail. The class directory referred to in the OSP handout is /home/course/cs3102/lab4

To begin, copy the following files from the class directory to your own working directory: -

Makefile, dialog.c, inter.c, procsvc.c and all sim.par* files (there are a total of 4 files)

You need to edit the files *.c to incorporate your own code. When you have edited the file, typing the Unix command make will compile your code and link them with the rest of the system, giving an executable file OSP.

3. Requirement.
The requirement is simple for this exercise. You should get no error and warning messages when you run your version of OSP against a set of provided simulation parameters. Your submission should have the following: -

a) program source (i.e. inter.c and procsvc.c)
b) test run results of the provided simulation parameters AND some your own parameters
c) a short lab report consisting of 2 parts: -
   - a flow chart showing control flow of interrupt handling in OSP
   - a discussion of the statistics obtained excluding memory utilization (you should vary the input parameters and study the effect on various statistics e.g. average wait time per process, CPU utilization etc.).

4. Submission.
The lab report and program source should be due by the end of week 10. Procedures of submission should be arranged between you and your lab tutor.

*** end ***
CS3102 Operating Systems
Laboratory V

1. Objective.

To explore further memory management concepts taught in lectures by implementing memory management functions and page replacement algorithms.

2. Activities.

In this laboratory session, you are going to implement one version out of four page replacement algorithms and related memory management functions in OSP.

You will write codes to satisfy the descriptions given in the OSP Programmer's Manual and the requirements in the next section, making use of what you have learned about operating systems and Lab 4 to fill in the missing details. The modules you write will be linked with the rest of OSP, and tested with various simulation parameters.

Before attempting to do this laboratory exercise, make sure that you have studied in detail sections 1.1 to 1.4, 1.6 and 2 of the OSP hand book: An Environment for Operating Systems Projects. The class account/directory referred to in the OSP pamphlet is /home/course/cs3102/lab6

To begin, copy the following files from the class directory to your own working directory: -

Makefile, memory.c, dialog.c

You need to edit the file memory.c to incorporate your own codes for all required internal routines. When you have edited the file, typing the Unix command make will compile your code and link them with the rest of the system, giving an executable file OSP.


Virtual memory and paging are used in OSP with page size = 512 bytes and a total of 32 frames as physical memory (these values can be changed by changing constants PAGE_SIZE and MAX_FRAME). The basic addressing scheme follows standard paging mechanism with each process having its own page table. The maximum logical address space is 16 pages.

You need to write codes to simulate hardware translation of addresses, i.e. look up of page table to get the frame number. Virtual address can be converted to a page table index by integer division of the virtual address by PAGE_SIZE. You need to keep track which pages are in memory and when a page not in memory is referred, a page fault interrupt has to be generated (how? refer Lab 4). You also need to maintain the status of frames, whether they are free or allocated, dirty or clean.

You are required to implement a version of replacement algorithm selected from the list below. Although it is not mandatory, it will be more interesting if you could coordinate with your classmates and compare simulation results among yourself by running them against matching sets of parameters. The 4 versions are :

(i) global replacement using LRU
(ii) local replacement using LRU
(iii) global replacement, replace first clean page first, then dirty page
(iv) local replacement, replace first clean page first, then first dirty page

You may use pure demand paging i.e. no pre-paging and no start cost calculation. When LRU is used as replacement algorithm, you need to design additional data structure in the frame table (by using hook to point to your own data structure).

Detail requirements for all internal routines can be found in OSP handbook and the following is a brief summary of how to implement them :

memory_init() Anything you wish to initialize when the system is booted. For LRU page replacement, this is the place you set up your own data structure.
prepage() Dummy return.
start_cost() deallocate() Use the input pcb to identify all frames and mark them as free.
lock_page() Use the input iorb to get the pcb and hence the page table, with the page_id, get to the frame, lock/unlock it.
unlock_page() refer() Use the input logic_addr to search page table, if page in memory, do whatever is necessary e.g. update time stamp, set dirty bit etc. If page is not in memory, raise a page fault interrupt.
get_page() Find a free frame, page in from disk and update page table. If no free frame is available, use replacement algorithm to get one, determine if page-out is necessary, page-in from disk, update page table.

4. Submission.

You will submit your programs with at least 4 simulation runs (preferably with additional files of yours) using the following simulation parameter files in

/home/course/cs3102/lab5 : -

sim.para1, sim.para2, sim.para3, sim.para4

You need to submit your program source, test results and a short report (max. 1 page) to compare the replacement algorithms based on the statistics obtained in your test runs and with statistics from at least one other classmate using a different replacement algorithm.
CS3102 Computer Operating Systems and Architecture
Laboratory VI

1. Objective.

To study the file system of UNIX and have hands-on exercises in manipulating files.

2. Activities I.

i) Use the man command, study the following frequently used file commands:
   
   ls, cp, in, rm, mv

   complete the following activities, you may need to find the commands to be used.

ii) Create a directory called class in your home directory.

iii) List descriptive information pertaining only to the class directory.

iv) Make class the current directory.

v) Under class, create two directories called subdir1 and subdir2.

vi) List recursively the contents of the class directory.

vii) Return to the home directory.

viii) Copy the file /home/course/cs3102/lab6.txt to your home directory having the name lab6.txt. Can you copy the file?

ix) Display the inode numbers of the original file and the copy from step (viii). Are the numbers the same or different? Why?

x) Link the lab6.txt as lab6.in in the directory subdir1. Are the inode numbers the same or different for these files? Why?

xi) Record the number of links pointing to the lab6.txt file.

xii) Move the lab6.in file in the directory subdir1 to subdir2 having the name lab6.mv. Are the inode numbers the same or different for these two files? Why?

xiii) List the content of subdir1 and subdir2 and record any files contained in each of these directories.

xiv) Return to your current directory. Recursively remove the class directory and its subdirectories and files requesting confirmation before each removal. List the contents of your home directory to verify that the class directory no longer exists.

3. Activities II.

i) Use man to study the command chmod, chown and umask.

ii) Create two short files called perm1 and perm2 using an editor. The files should contain one or two lines of text each.

iii) Display the permission mode for these files. Record the commands used and the permission mode for each file.

iv) Change the access mode for perm1 using octal format so that the owner can access and modify the file, group members can only view the file, and all other users are completely restricted from accessing the file. Record the command used. Display the permissions for perm1 to confirm that the new access mode was changed correctly.

v) Change the access mode for perm2 using symbolic format to remove all access by group members and all other users (except the owner). Record the command used. Display the permissions for perm2 to confirm that the new access mode was changed correctly.

vi) Display and record the current umask setting.

vii) Change the default permission to restrict all access by group members and other users for new files and directories. Record the command used.

viii) Create the file perm3 containing one or two lines of text using an editor.

ix) Display and record the permission mode for perm3. What effect does the current default permission have on the three files perm1, perm2 and perm3? Explain.

x) Change the mode for perm1 and perm2 to read and write for all users. Confirm the change.

xi) Change and record the ownership of perm3 to you neighbour. Confirm this change. Change the ownership of perm3 back to your login name. Record the result. Explain.