

Operating Systems exercises – week 8 – Selected answers.

Tutorial questions

Longer questions.

1) Consider a paging system with the page table stored in memory.

- a. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?
- b. If we add associative registers, and 75 percent of all page-table references are found in the associative registers, what is the effective memory reference time? (Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there.)

Answer:

- a. 400 nanoseconds; 200 nanoseconds to access the page table and 200 nanoseconds to access the word in memory.
- b. Effective access time = $0.75 * (200 \text{ nanoseconds}) + 0.25 * (400 \text{ nanoseconds}) = 250 \text{ nanoseconds}$.

2) When do page faults occur? Describe the actions taken by the operating system when a page fault occurs.

Answer:

A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid a free frame is located and I/O requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

3) Consider a demand-paging system with the following time-measured utilizations:

CPU utilization 20%

Paging disk 97.7%

Other I/O devices 5%

Which (if any) of the following will (probably) improve CPU utilization? Explain your answer.

- a. Install a faster CPU.
- b. Install a bigger paging disk.
- c. Increase the degree of multiprogramming.
- d. Decrease the degree of multiprogramming.
- e. Install more main memory.
- f. Install a faster hard disk, or multiple controllers with multiple hard disks.
- g. Add prepaging to the page fetch algorithms.
- h. Increase the page size.

Answer:

The system obviously is spending most of its time paging, indicating over-allocation of memory. If the level of multiprogramming is reduced resident processes would page fault less frequently and the CPU utilization would improve. Another way to improve performance would be to get more physical memory or a faster paging disk.

- a. Get a faster CPU —No.
- b. Get a bigger paging disk —No.
- c. Increase the degree of multiprogramming — No.
- d. Decrease the degree of multiprogramming —Yes.
- e. Install more main memory — Likely to improve CPU utilization as more pages can remain resident and not require paging to or from the disks.
- f. Install a faster hard disk, or multiple controllers with multiple hard disks — Also an improvement, as the disk bottleneck is removed by faster response and more throughput to the disks, the CPU will get more data more quickly.
- g. Add prepaging to the page fetch algorithms — Again, the CPU will get more data faster, so it will be more in use. This is only the case if the paging action is amenable to prefetching (i.e., some of the access is sequential).
- h. Increase the page size — Increasing the page size will result in fewer page faults if data is being accessed sequentially. If data access is more or less random, more paging action could ensue because fewer pages can be kept in memory and more data is transferred per page fault. So this change is as likely to decrease utilization as it is to increase it.

4) Consider the following page reference string:

1, 2, 3, 2, 4, 2, 5, 2, 3, 6, 2, 5, 1, 2, 3

How many page faults would occur for the following replacement algorithms, assuming three and four frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

- LRU replacement
- FIFO replacement
- Optimal replacement

No answer – question and answer to be included in portfolio that is to be handed in.

5) Consider a demand-paging system with a paging disk that has an average access and transfer time of 20 milliseconds. Addresses are translated through a page table in main memory, with an access time of 1 microsecond per memory access. Thus, each memory reference through the page table takes two accesses. To improve this time, we have added an associative memory that reduces access time to one memory reference, if the page-table entry is in the associative memory.

Assume that 80 percent of the accesses are in the associative memory, and that, of the

remaining, 10 percent (or 2 percent of the total) cause page faults. What is the effective memory access time?

Answer:

effective access time = $(0.8) * (1 \text{ microsec})$
+ $(0.18) * (2 \text{ microsec}) + (0.02) * (20002 \text{ microsec})$
= 401.2 microsec
= 0.4 millisec

6) Consider a demand-paged computer system where the degree of multiprogramming is currently fixed at four. The system was recently measured to determine utilization of CPU and the paging disk. The results are one of the following alternatives. For each case, what is happening? Can the degree of multiprogramming be increased to increase the CPU utilization? Is the paging helping?

- a. CPU utilization 13 percent; disk utilization 97 percent
- b. CPU utilization 87 percent; disk utilization 3 percent
- c. CPU utilization 13 percent; disk utilization 3 percent

Answer:

- a. Thrashing is occurring.
- b. CPU utilization is sufficiently high to leave things alone. Can increase degree of multiprogramming.
- c. Increase the degree of multiprogramming.

Shorter questions.

1) What is virtual memory?

Answer:

A set of techniques and hardware allowing us to execute a program even when not entirely in memory.

2) What is demand paging?

Answer:

Design where a page is never swapped into memory unless needed.

3) What is page replacement?

Answer:

Selecting a frame (preferably not in use) as a victim; swap it out; swap in the desired page into this frame; restarting program.

4) What is Belady's anomaly?

Answer:

You'd expect that the page-fault rate would decrease as the number of frames increases; but Belady's anomaly says this is not true in all algorithms.

5) What is thrashing?

Answer:

State where the system spends an excessive amount of time on paging, compared to the execution of processes.

Windows process creation and virtual memory exercises.

1) Start the Windows virtual machine [Start, then VMware, then VMware Player, and select Windows XP virtual machine]

Remember to press CTRL+G to direct input to the virtual machine and CTRL+ALT to direct it to the host computer.

2) We are going to do some more Windows system programming. We will be using the development environment we used before called Dev C/C++.

So start Dev C/C++, go to Start and you will find an icon for Dev C/C++ in the most recently used list of programs. Click on it.

3) This will bring up Dev C/C++. Close the Tip of the Day. You will now see the interface for Dev C/C++.

4) Click on File menu then New and then Source File. This will then bring up a text editor screen where previously it was greyed out. You can enter program code in that text area. Entry is normal and the editor responds to cursor keys and the mouse as you would expect.

5) We will be using the following Windows system call:

CreateProcess – this as the name suggests creates a new process

It needs the include file <windows.h>.

```
#include <windows.h>
```

It returns – zero if it fails and nonzero value if it succeeds

It has 10 parameters (in order):

- i) a string representation of the name of the executable file – usually set to NULL as
- ii) is the preferred method of invoking the new process
- ii) a string representation of the command line that would run the process from the command line
- iii) a security attribute parameter specifying process security attributes – use default value of NULL
- iv) a security attribute parameter specifying thread security attributes – use default value of NULL
- v) a Boolean that sets whether or not the created process is given its own handles to the objects/resources owned by the creating process – use FALSE
- vi) a set of flags that determine the priority level of the created process –
NORMAL_PRIORITY_CLASS is the default. We could also make the new process run in its own console window by ORing priority class with
CREATE_NEW_CONSOLE

- vii) a parameter that points to a block of strings that specify any changes to environment variables. NULL means the created process uses the same environment variable values as the creating process
- viii) a string that specifies the drive and directory pathname of the directory in which the created process should run. NULL means the process executes in the working directory of the parent process
- ix) a pointer to a struct with a set of startup information
- x) a pointer to a struct of process information i.e. pass the address of a struct into which the process information will be placed – the struct includes the process id number of the created process and the thread id number of the base thread of the process, and 2 handles to new process and the base thread. Before exiting we will need to close the handles to process and thread.

So template code to create a process could be,

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

int main (int argc, LPTSTR argv [])
{

    STARTUPINFO startinfo;
    PROCESS_INFORMATION processinfo;
    LPTSTR lpCommandLine;

    ZeroMemory(&startinfo, sizeof(startinfo));
    startinfo.cb = sizeof(startinfo);
    strcpy(lpCommandLine, "c:\\WINDOWS\\SYSTEM32\\NOTEPAD.EXE temp.txt");

    if (!CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
        NORMAL_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, NULL,
        &startinfo, &processinfo)) {
        printf("CreateProcess failed on error %d\n", GetLastError());
        ExitProcess(1);
    }

    CloseHandle(&processinfo.hThread);
    CloseHandle(&processinfo.hProcess);

}
```

This program will create a process that will execute Notepad with a file temp.txt.

6) Click on File menu then New and then Source File. This will then bring up a text editor screen where previously it was greyed out. You can enter program code in that text area.

Enter the code above.

7) Now save it. When you are prompted for a location My Documents comes up by default. We want to save it to a different location. Go to C:drive, then Documents and Settings and then user1. You must give the program a name. We will call it newProcess.c and change the Save File as Type to C source files (.c).

Save it.

8) Compile the program we have typed in - go to menu Execute and select Compile and deal with any compiler errors (shown below the editing pane).

9) Run the program from the command line (Start, then Run and type cmd.exe to bring up the command line window.

If you type at the prompt,

```
dir
```

This is the directory listing command. You should find newProcess.c and newProcess.exe listed.

10) Now you can run the program as

```
newProcess
```

This should give you a copy of Notepad with temp.txt running from the Documents and Settings\user1 directory.

11) Enter some text in temp.txt and save it. Close Notepad. Use dir to look at the user1 directory and you should find the file temp.txt.

12) Run the program newProcess again.

13) Start the Task Manager by pressing Ctrl+Shift+Esc.

You can look at the Applications that are running and you can also see the processes. You can identify Notepad in the applications and you should also be able to see the Notepad process when you look at the processes. However, you cannot see the parent process because it terminated as soon as it created the new process.

14) In order for us to see it we can make it wait for the child process to terminate before exiting itself.

We can use the system call WaitForSingleObject. This waits until a synchronization object is signaled. A process handle is set to signaled when it terminates.

The call takes 2 parameters, a handle to a synchronization object (we will use the process handle) and a timeout value, which allows the process to proceed after

waiting for the timeout length of time (in milliseconds). INFINITE means it will wait indefinitely for the process to terminate.

So if we add the following to our program (after the close handle code), the parent process will wait until the child process terminates.

```
WaitForSingleObject(processinfo.hProcess, INFINITE);
```

// ANSWER to question 14

```
#include <windows.h>
#include <stdio.h>
#include <string.h>

int main (int argc, LPTSTR argv [])
{

    STARTUPINFO startinfo;
    PROCESS_INFORMATION processinfo;
    LPTSTR lpCommandLine;

    ZeroMemory(&startinfo, sizeof(startinfo));
    startinfo.cb = sizeof(startinfo);
    strcpy(lpCommandLine, "c:\\WINDOWS\\SYSTEM32\\NOTEPAD.EXE temp.txt");

    if (!CreateProcess(NULL, lpCommandLine, NULL, NULL, FALSE,
        NORMAL_PRIORITY_CLASS | CREATE_NEW_CONSOLE, NULL, NULL,
        &startinfo, &processinfo)) {
        printf("CreateProcess failed on error %d\n", GetLastError());
        ExitProcess(1);
    }

    CloseHandle(&processinfo.hThread);
    CloseHandle(&processinfo.hProcess);

    WaitForSingleObject(processinfo.hProcess, INFINITE);

}
```

15) Save and recompile the program and run it again. Now look at Task Manager again. This time as well as the process Notepad.exe you should also be able to see newProcess.exe.

16) Write another program that outputs, using an infinite loop, a series of even (pair) numbers. Save and compile this program. Call it even.c.

Now modify the original program so that it invokes the program even.exe, rather than NotePad.exe.

17) Run the new version of newProcess.exe. Now you should see the numbers being output, but the output is in a different command line window to the parent process.

18) If you check Task Manager you will see the even.exe process running as well as newProcess.exe.

// ANSWER to question 16

```
#include <windows.h>
```

```
#include <stdio.h>
```

```
int main (int argc, LPTSTR argv [])
```

```
{
```

```
int i;
```

```
    for (i=2;;i=i+2)
```

```
        printf("%d\n",i);
```

```
}
```

// PLUS this mod to q14 program

```
strcpy(lpCommandLine, "c:\\Documents and Settings\\user1\\even.exe");
```