Homework 1 Solutions
Operating Systems, V22.0202
Fall 2006, Professor Yap

Due: Wed Sep 20, in class
SOLUTION PREPARED BY Instructor and T.A.s

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.

---

1. (2 Points each)

(a) Exercise 1.1 (p.26). What are the two main purposes of an OS?

(b) Exercise 1.3 (p.26). What are the main advantages of multiprogramming?

(c) Exercise 2.9 (p.56). Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large a the device for which it is caching (e.g., a cache as large as a disk), why not make it that large and eliminate the device)?

(d) Exercise 3.6 (p.98). List five services provided by an OS. Explain how each provides convenience to the users. In what cases would it be impossible for user-level programs to provide these services? Explain.

SOLUTION

(a) What are two main functions of an OS?

1. To provide a multiprogramming environment for running processes. This means timesharing and a virtual environment for each process. This is described in Section 1.1.1 (The OS as an Extended Machine)

2. To manage the resources which are shared by processes. Section 1.1.2 (The OS as a Resource Manager).

3. Closely related to 1., but with a different emphasis, you can say that an OS provides an extended (virtual) machine environment for processes – this includes provide conveniences and services that are enhancements of the raw hardware. The OS offers such services via system calls.

(b) What are the main advantages of multiprogramming?

1. Several processes can share a common hardware environment (CPU, disk, printer, etc) concurrently, while each acts independently or obliviously of the others.

2. Resources are utilized more efficiently. For instance, if several processes are concurrent, then an I/O-bound process would not prevent other CPU-bound processes from running.

(c) Why are caches useful, etc.

Reason 1: caches are useful when two or more components need to exchange data, and the components perform transfers at different speeds. E.g., the CPU registers and the disk are two such components. Caches solve this problem by providing a buffer of intermediate speed to store part of the data in the slow component. If the fast component finds the data it needs in the cache, it need not wait for the slower component. The data in the cache must be kept consistent with the data

Reason 2: cache is used in prefetching of instructions in the CPU execution cycle – we introduce a cache to hold the next few instructions, so that we do not have to wait for slow memory access. In this case, the cache is more like a pipeline.

Problems created by caching: since cache hold temporary information for a slower memory component, we need to maintain consistency between the copy in the cache with its actual value in the memory component. This requires extra checking when writing to memory, etc.

Why not just replace the slow component by a large cache? First problem is economic – cache memory is usually much more expensive than the slower memory that it is a surrogate for. Second is difference in technology – usually cache memory is dynamic (it does not retain its information when power is turned off, and thus it cannot be a substitute for permanent memory such as a disk).

(d) List five OS services, etc.

HINT: To answer this question, think of the system calls an OS provides.

1. Program execution. ("exec") The OS loads the executable file into memory and starts execution. [In multiprogramming, we need to schedule the processes, and only an impartial OS can be trusted to do this correctly.]

2. I/O operations. ("read/write") These operations require very low level communication and control. The OS shields the users from having to know these details. [User programs may cause damage to the file systems with improper operation.]

3. File Manipulation ("creation", "deletion", "allocation", "renaming", etc) [We need protection and enforcement of file access previledges – these cannot be left to user programs.]

4. Communication. ("send", "receive", etc) This requires low level protocols. [Again, the integrity of the communication system requires kernel protection]

5. Error Handling. Data transfers may be imperfect, so error needs to be detected and corrected. Files need to be checked after a reboot, etc. [All these require system-wide knowledge, and such knowledge is often beyond any user's competence]

♠

2. (10 Points)
An alert reviewer notices a consistent spelling error in the manuscript on Operating Systems textbook, about to go to press. The book has about 700 pages, each with 50 lines of 80 characters each. How long would it take to process (i.e., make one pass through) the manuscript, assuming that the manuscript is in the following levels of the memory hierarchy:

	TYPE	CAPACITY	ACCESS TIME
(a)	Registers	< 1KB	1 nsec
(b)	Cache	1 MB	2 nsec
(c)	Main Memory	1 GB	10 nsec
(d)	Magnetic Disk	50 GB	10 msec
(e)	Magnetic Tape	100 GB	100 sec

For internal storage methods, assume the access time is per character, for disk devices, assume the access time is per block of 1024 characters, and for tape assume the access time is to the time to get to the start of the data, with subsequent access time similar to disk access.

SOLUTION Solution:

There file is 700x50x80 = 2.8MB bytes large.

1. There is no way it could fit into the registers of a machine.

2. Cache. It could fit into the cache of a big machine. The time to read this file at 2 ns./byte would take

$$(2 \times 10^{-9} \times (2.8 \times 10^6) = 5.6 \times 10^{-3} = 5.6ms.$$

3. Main Memory. This is 10 ns/byte and so is 5 times slower, at 28 ms.

4. Disk. Each block is 1024 Bytes, so the file takes 2800 blocks. Time to access each block is $10ms$, so the total time is $10 \times 10^{-3} \times 2800 = 28sec$.

5. Tape. We need 100 sec to access the tape, and thereafter, it has the same speed as the disk. Thus the time is $128sec$. ♠

3. (62 Points)

This question assumes that you have downloaded Cygwin on your computer. Also, you have downloaded gcc, make and tar programs. Please look at our class links for this information.

Enclosed is a simple program called `fork.c` and `test.c` that calls fork() and exec*(). They are put into a tar file called "homework1.tar". You are to write a variation of these programs.

- 0. Read our quick introductions to Cygwin, C programming and Make program. These are links under RESOURCES on class page.
- 1. Untar the file homework1.tar (move the file to a directory and type "tar -xvf homework1.tar"). Among the files, you will see "fork.c" and "Makefile". Please compile fork.c by typing "make". If gcc compiler creates an executable file called "a.out", you can execute the program by typing "a.out". [Note: a.out is the standard compiler output on UNIX]
- 2. Please create a variation of the fork.c program which will create an arbitrary number of processes until your system crashes (there might be some other possibilities, depending on your system).

  BE SURE THAT ALL YOUR PROCESSES ARE KEPT AROUND FOR THE DURATION OF THE EXECUTION, NOT JUST CREATED TO DIE IMMEDIATELY. For safety, we suggest stopping all other programs before you run the test.
- 3. There is a simple make program also provided here. You must use this to help you programming and testing.
- 4. Read the following submission rules carefully, as up to 15 percent of this question may be deducted for non-compliance.

  – Make sure that your program is well-documented (long and short comments)

  – Make sure your name, contact information, and any other information you like, is given in a README file.

  – ANY ACKNOWLEDGEMENT OF SOURCES SHOULD BE EXPLICITLY MADE HERE. Otherwise, it is considered plagiarism.

  – Your C program file name MUST be called h1.c.

  – You must put h1.c, README, Makefile, and any other necessary files into a tar file called h1.tar. (Type "tar cvf h1.tar h1.c Makefile ..." to create this tar file).

  – Send the tar file to the grader (but cc to me). After we untar your file, we expect to type "make" to compile it, and to type "make test" to run the compiled program.

SOLUTION

We grade your conformance with the instructions above.

In particular, we want you to keep the processes around after creation. To do this, you would need to call "exec" to sleep (as in the sample program).

It would be nice if you also make each process announce its presence to the world (echo or printf).

♠

```
// file: test.c
// Comp.Sys.Org.II, Spring 2005, Yap.
// The following code is a simple modification of the "fork.c" code from
//
// Dr Ian G Graham, Department of Information Technology, GUGC
// Copyright  2000-2003
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>

extern int errno;     /* system error number */

void syserr(char* );  /* error report, abort routine */

int main(int argc, char *argv[])
{
   pid_t pid;
   int rc;

   pid = getpid();
   printf("Process ID before fork: %d\n", pid);

   switch (fork()) {
      case -1:
         syserr("fork");
      case 0:

/* execution in child process */
         pid = getpid();
         printf("Process ID in child after fork: %d\n", pid);
         execlp("echo", "0", "Child to world: Hello!",NULL);
         syserr("execl");
   }

/* continued execution in parent process */

   pid = getpid();
   printf("Process ID in parent after fork: %d\n", pid);

   //getchar(); // hang until I type something (alternatively,
    // let parent wait for return of child process.
   exit(0);
}

void syserr(char * msg)
{
   fprintf(stderr,"%s: %s", strerror(errno), msg);
   //abort(errno); // variant
   abort();
}
```