

Northern Michigan University (Marquette Co, MI)

CS426-01-26F: Operating Systems (Andrew A. Poe)
Midterm Examination (Exam 1) Page 1/2

Name: _____
Friday 9 October 2026 3:00 P.M. EDT

Time: 50 minutes.

1. Using only low-level system calls, write the following program in C: The program is called `printit`. It prints the string residing in the first argument. The second argument is an integer. If that integer is greater than one, the program should fork, and the child should run `printit` with the same string but with the integer decremented by one. The parent has nothing else to do, but there should be no zombies at any point. For example, `./printit "Hello" 5`, should print "Hello" to the screen, then fork, and the child would then run `./printit "Hello" 4`.

`strlen(s)` gets the length of a string. `atoi(s)` returns the integer equivalent of a string. `atoi("5")` returns 5.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

int main (int argc, char **argv) {

    int n = atoi (argv[2]);
    write (1,argv[1],strlen(argv[1]));
    if (n > 1) {
        int pid=fork();
        if (pid==0) {
            char nstr[50];
            sprintf (nstr,"%d",n-1);
            execl ("./printit","printit",nstr);
        }
        wait (NULL);
    }
    return 0;
}
```

2. In C, write a complete program, using only low-level system calls. The program should fork, read exactly five bytes from standard input and pass those five bytes through a pipe to the child. The child should open the file specified in the first argument for writing, redirect that file to standard output, then read the five bytes from the pipe and write them to the (new) standard output. For example, `./prob2 stuff.txt` : parent reads five bytes from keyboard or whatever, writes them to a pipe. Child reads from the pipe and writes them to `stuff.txt` AFTER redirecting standard output to `stuff.txt`. Both parent and child should terminate without creating a zombie.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <pipe.h>
#include <sys/wait.h>
```

Northern Michigan University (Marquette Co, MI)

CS426-01-26F: Operating Systems (Andrew A. Poe)
Midterm Examination (Exam 1) Page 2/2

Name: _____
Friday 9 October 2026 3:00 P.M. EDT

```
int main (int argc, char **argv) {  
  
    int p[2];  
    pipe (p);  
    int pid = fork();  
    if (pid==0) {  
        close (p[1]);  
        int fd = open (argv[1], O_WRONLY|O_CREAT|O_TRUNC, 0700);  
        dup2 (fd, 1);  
        close (fd);  
        char buf[5];  
        read (p[0], buf, 5);  
        write (1, buf, 5);  
        close (p[0]);  
    } else {  
        close (p[0]);  
        char buf[5];  
        read (0, buf, 5);  
        write (p[1], buf, 5);  
        close (p[1]);  
        wait (NULL);  
    }  
}
```

3. If I have a memory space of 25 bytes, initially clear, what will the results be with this sequence of commands under Best Fit?

alloc 5	Used:<0,4> Free:<6,24> Returns 0
alloc 5	Used:<0,4>,<5,9> Free:<11,24> Returns 5
alloc 5	Used:<0,4>,<5,9>,<10,14> Free: <15,24> Returns 10
alloc 2	Used:<0,4>,<5,9>,<10,14>,<15,16> Free: <17,24> Returns 10
dealloc 5	Used:<0,4>,<10,14>,<15,16> Free: <5,9>,<17,24> Returns 5
alloc 2	Used:<0,4>,<5,6>,<10,14>,<15,16> Free: <7,9>,<17,24> Returns 5

4. Every operating system I've ever worked with offers low-level commands to add new bytes to the end of a file, delete bytes from the end of a file, and to overwrite existing characters with new characters. I've never encountered an OS that offers to low-level commands to insert bytes between existing bytes or to shorten a file by removing internal bytes. These features would certainly be useful so what would be the problem in implementing them? Design a system that might work for these features and explain the difficulties in your system.

The only way to physically insert bytes on a disc is to move the bytes that were already there someplace else and this could be a slow process, particularly on a large file where a large portion of the file may have to be relocated.

One way to do it would be a variant a B-tree with order statistics. Break a file up into blocks which do not have to be completely filled. When stuff gets inserted or deleted then only one block will have to be partially rewritten, not the whole file. If a block gets too

Northern Michigan University (Marquette Co, MI)

CS426-01-26F: Operating Systems (Andrew A. Poe)
Midterm Examination (Exam 1) Page 3/2

Name: _____
Friday 9 October 2026 3:00 P.M. EDT

large it can be split into two, if a block gets too small, it could be coalesced with a neighboring block. And the tree structure of these blocks would allow random access with reasonable efficiency. The problem is that this is a very complex data structure with more overhead than you would probably want.