assignment 1

cs 540 operating systems

morgan bergen

due august 13 2024

to succeed in this assignment, prioritize conciseness - convey your ideas in a few words as possible. embrace open-ended questions, where there's no single correct answer. instead make reasonable assumptions and respond as a wireless designer or researcher would. each question has a value of 1 pt.

1. how does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) system?

kernel mode is a privileged mode of operating where the operating system has unrestricted access to all hardware and system resources, as opposed to user mode where more resources are restricted. in kernel mode, the operating system can execute any cpu instruction and reference any memory address while user mode applications have limited access to system resources and cannot directly interact with hardware or reference arbitrary memory addresses.

the distinction between kernel mode and user mode acts as a protection mechanism in terms of separation, controlled access, and fault isolation. separation allows for protection by separating user applications from critical system operations, controlled access ensures that user applications must be required to make system calls to access hardware or perform privileged operations, and fault isolation ensures that if a user application crashes or behaves maliciously, the damage is contained within the user mode, preventing it from affecting the overall system stability.

2. what is the purpose of system calls?

the purpose of a system call is to provide a controlled interface for user applications to request services from the operating system's kernel. system calls allow user

applications to access low level services provided by the operating system, such as process control, file manipulation, device management, information management, communication, and protection.
some examples of unix commands are `fork()`, `open()`, `read()`, `getpid()`, `pipe()`, `chmod()`.

3. describe the actions taken by a kernel to context-switch between processes

1. saves the state of the old process to the process control block pcb
2. then loads the saved state from the process control block for the new process

4. what are common commands and their respective functionalities in the unix/linux command shell, and how can they be used effectively to manage files and directories? please list at least five of them

`ls` - list directory contents

usage `ls [options] [directory]`

options - `-r` - reverse sort order, `-s` - display size of each file in blocks, `-t` - sort by modification time, `-u` - access time, `-v` - print raw characters, `-v -w` - print raw characters

example

```
❯ ls -l .
total 8
-rw-r--r--@ 1 owner  staff  1678 Sep  3 12:31 README.md
drwxr-xr-x  5 owner  staff   160 Sep  5 13:23 materials
drwxr-xr-x  6 owner  staff   192 Sep 12 13:46 notes
```

by listing files and directories, users can easily navigate and manage their filesystem within the path provided. options such as `-l` provide detailed information including file permissions, owner, size, and modification date.

`mkdir` - create directory

usage - `mkdir [options] directory_name`

options - `-m` set permissions, `-p` make parent directories, `-v` print message for each created directory

example

```
~/Documents/computer-ethics main*
❯ mkdir temp

~/Documents/computer-ethics main*
❯ ls
total 24
drwxr-xr-x   9 owner  staff   288B Sep 12 15:21 .
drwx------@ 21 owner  staff   672B Sep 12 15:20 ..
-rw-r--r--@  1 owner  staff   6.0K Aug 20 12:44 .DS_Store
drwxr-xr-x  15 owner  staff   480B Sep 12 15:18 .git
drwxr-xr-x@  3 owner  staff    96B Sep 12 11:32 .vscode
-rw-r--r--@  1 owner  staff   1.6K Sep  3 12:31 README.md
drwxr-xr-x   5 owner  staff   160B Sep  5 13:23 materials
drwxr-xr-x   6 owner  staff   192B Sep 12 13:46 notes
drwxr-xr-x   2 owner  staff    64B Sep 12 15:21 temp
```

`rmdir` - remove directory

usage - `rmdir [options] directory_name`

options - `-p` - remove parent directories if empty, `-v` - print message for each removed directory

example

```
~/Documents/computer-ethics main*
❯ rmdir temp

~/Documents/computer-ethics main*
❯ ls
total 24
drwxr-xr-x   8 owner  staff   256B Sep 12 15:29 .
drwx------@ 21 owner  staff   672B Sep 12 15:29 ..
-rw-r--r--@  1 owner  staff   6.0K Aug 20 12:44 .DS_Store
drwxr-xr-x  15 owner  staff   480B Sep 12 15:18 .git
drwxr-xr-x@  3 owner  staff    96B Sep 12 11:32 .vscode
-rw-r--r--@  1 owner  staff   1.6K Sep  3 12:31 README.md
drwxr-xr-x   5 owner  staff   160B Sep  5 13:23 materials
drwxr-xr-x   6 owner  staff   192B Sep 12 13:46 notes
```

`mv` - move file

usage - `mv [options] source destination`

options - `-i` - prompt before overwrite, `-v` - print message for each file moved

example

```
~/Documents/computer-ethics main*
❯ mv notes ../../Documents/
```

`pwd` - print working directory

usage - `pwd`

example

```
~/Documents/computer-ethics main*
❯ pwd
/Users/owner/Documents/computer-ethics
```

5. consider the following c code snippet that demonstrates the use of the fork system call, and answer the following questions

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

int main() {

    pid_t pid = fork();

    if (pid == -1) {
        //  error handling
        perror("fork");
        return 1;

    } else if {
        // child process
        printf("child process:  my pid is %d\n", getpid());

    } else {
        //  parent process
        printf("parent process:  my pid is %d\n", getpid());
        printf("parent process:  child pid is %d\n", pid);
    }
```

```
    return 0;
}
```

a - explain the purpose of the fork system call in this code. how does it facilitate the creation of a new process?

the fork system call is used to create a new process by duplicating the calling process. this new process is called the child process, while the original process is called the parent process. the new process (child) is an exact copy of the calling process (parent) except for the returned value.

b - describe the behavior of the child process and the parent process after the fork call. what information does each process print?

the child process will print the message "child process: my pid is [pid]" and the parent process will print the message "parent process: my pid is [pid]" and "parent process: child pid is [pid of child process]". pid is the process id of the calling process.

c - what are the potential return values of the fork system call? how does the code handle each possible scenario?

if the `fork()` returns a negative value it indicates that the creation of the child process was not successful. if `for()` turns zero, it indicates that the code is running the child process. if the `fork()` returns a positive value, it indicates that the code is running in the parent process, and the value is the pid of the newly created child process.

d - discuss the concept of copy on write memory as it related to the memory management of the child process. how does it optimize memory usage in this context?

copy on write memory is a memory management technique where the parent and child process initially share the same memory pages. when either the parent or child process attempts to modify a page, the operating system creates a copy of that page with the changes, and both the parent and child processes and have their own copy of the modified page. this allows for memory optimization by avoiding the need to duplicate the entire memory space of the parent process for the child process, only copying pages that are modified, and allows for both processes to have their own copy of the modified page.

e - how are file descriptors handled between the parent and child processes? are

they shared or copied? explain your answer.

file descriptors are not shared between the parent and child processes. each process has its own set of file descriptors. when a child process is created, it inherits the file descriptors of the parent process. if the child process needs to access a file that is open in the parent process, the child process must duplicate the file descriptor using the `dup2()` system call.

f - what potential issues or considerations should be taken into account when managing shared resources between parent and child processes in a multi-process environment?

when managing shared resources between parent and child processes in a multi-process environment, potential issues or considerations that should be taken into account include:

1. thread cancellation - is the process of terminating a thread before it has completed its execution. the potential issue is that the thread may be in the middle of critical section of code and may not complete. the solution is to use synchronization mechanisms to ensure that the thread completes its critical section of code before it is terminated.

2. signal handling - is the process of handling signals sent to a thread. the potential issue is that the thread may be in the middle of critical section of code and may not complete. the solution is to use synchronization mechanisms to ensure that the thread completes its critical section of code before it is terminated.

3. handling thread specific data - is the process of handling thread specific data. the potential issue is that the thread may be in the middle of critical section of code and may not complete. the solution is to use synchronization mechanisms to ensure that the thread completes its critical section of code before it is terminated.