EE 3233 System Programming for Engineers - Summer 2025

Exam 2

(Monday, July 14)

Name: Arnav Gupta Score: /130

Total 6 pages

[1 - 3] Refer to the following bash command results and answer the questions:

```
$ ls -al
-rw-rw-r-- 1 user1 staff    0 Nov 17 20:29 2.py
-rw-rw-r-- 1 user1 staff    0 Nov 17 20:30 foo.txt
-rw-rw-r-- 1 user1 staff    0 Nov 17 20:30 func1.py
-rw-rw-r-- 1 user1 staff    0 Nov 17 20:30 func2.py
drwxrwxr-x 2 user1 staff 4096 Nov 17 20:30 myDir

$ ls -al myDir
-rw-rw-r-- 1 user1 staff    0 Nov 17 20:32 abc.txt

$ grep -r 123456 *
2.py:123456
func2.py: func2(., 123456)
myDir/abc.txt:123456
(grep is a command to search a string, -r indicates recursive search)
```

1. What will be the result when you run the following python code (func1.py) with the command? **`$ python3 func1.py .`**

```python
#!/usr/bin/python3
import os
import sys


def func1(startdir):
    for (thisDir, dirsHere, filesHere) in os.walk(startdir):
        for fname in filesHere:
            fpath = os.path.join(thisDir, fname)
            print(fpath)


if __name__ == '__main__':
    func1(sys.argv[1])
```

☐ a. ./func1.py
./2.py
./foo.txt
./func2.py

☑ b. ./func1.py
./2.py
./foo.txt
./func2.py
./myDir/abc.txt

☐ c. ./2.py
./func2.py
./myDir/abc.txt

☐ d. None of them

2. What will be the result when you run the following python code (func2.py)?
$ python3 func2.py

```
#!/usr/bin/python3
import os


def func2(startdir, strToFind):
    for (thisDir, dirsHere, filesHere) in os.walk(startdir):
        for fname in filesHere:
            fpath = os.path.join(thisDir, fname)
            if strToFind in open(fpath).read():
                print(fpath)


if __name__ == '__main__':
    func2(".", "123456")
```

☐ a. ./func1.py
./2.py
./foo.txt
./func2.py

☐ b. ./func1.py
./2.py
./foo.txt
./func2.py
./myDir/abc.txt

☑ c. ./2.py
./func2.py
./myDir/abc.txt

☐ d. None of them

3. If you run the following python code (func3.py), what will be the result? **python3 func3.py**

```
#!/usr/bin/python
import fnmatch
import os


startdir = "."
for (thisDir, dirsHere, filesHere) in os.walk(startdir):
    for fname in dirsHere + filesHere:
        if fnmatch.fnmatch(fname, "*.txt"):
            print(fname)
```

☐ a. ./foo.txt
./myDir/abc.txt

☑ b. foo.txt
abc.txt

☐ c. ./foo.txt

☐ d. ./myDir/abc.txt

4. When you run the following C code (**t_kill**), choose a CORRECT statement about the outcome?

```c
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

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

    if (argc != 3 || strcmp(argv[1], "--help") == 0) {
        printf("%s pid sig-num\n", argv[0]);
        exit(-1);
    }

    sig = atoi(argv[2]);
    pid = atoi(argv[1]);
    s = kill(pid, sig);

    if (sig != 0) {
        if (s == -1) {
            printf("kill\n");
            exit(-1);
        }
    }
    else { /* Null signal: process existence check */
        if (s == 0) {
            printf("Process exists and we can send it a signal\n");
        }
        else {
            if (errno == EPERM) {
                printf("Process exists, but we don't have "
                       "permission to send it a signal\n");
            }
            else if (errno == ESRCH) {
                printf("Process does not exist\n");
            }
            Else {
                printf("kill\n");
                exit(-1);
            }
        }
    }

    exit(EXIT_SUCCESS);
}
```

☑ a. When a process with PID=3429 for example, is running, if you run the t_kill as **`$ ./t_kill 3429 0,`** it will give you message, "Process exists and we can send it a signal"

☐ b. When a process with PID=3429 for example, is running, if you run the t_kill as **`$ ./t_kill 3429 0, i`**t will terminate the process 3429

☐ c. When a process with PID=100 for example, is running, if you run the t_kill as **`$ ./t_kill 100 0`**, it will exit with an error message, "kill"

☐ d. When a process with PID=100 for example, is running, if you run the t_kill as **`$ ./t_kill 100 0`**, it will print out "Process does not exist"

5. Which code section will be not interrupted by SIGINT?

☐ a. A ☐ b. B ☑ c. C ☐ d. D

```
sigset_t blockSet, prevMask;
```

A

```
sigemptyset(&blockSet);
sigaddset(&blockSet, SIGINT);
```

B

```
if (sigprocmask(SIG_BLOCK, &blockSet, &prevMask) == -1)
    errExit("sigprocmask1");
```

C

```
if (sigprocmask(SIG_SETMASK, &prevMask, NULL) == -1)
    errExit("sigprocmask2");
```

D

6. nice values are inherited by child processes?

☑ a. True ☐ b. False

7. What do you expect when you run the following code assuming that child PID=71, parent PID=70?

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

static int idata = 100;
int main(int argc, char *argv[])
{
    int istack = 200;
    pid_t childPid;

    switch (childPid = fork())
    {
    case -1:
        printf("error: fork\n");
        exit(-1);
    case 0:
        idata++;
        istack++;
        break;
    default:
        sleep(2);
        idata *= 2;
        istack *= 2;
        break;
    }
    printf("PID = %ld %s idata = %d istack = %d\n", (long)getpid(),
           (childPid == 0) ? "(child) " : "(parent)", idata, istack);
    exit(EXIT_SUCCESS);
}
```

☐ a. PID=71 (child) idata=200 istack=400
PID=70 (parent) idata=101 istack=201

☐ b. PID=71 (child) idata=0 istack=0
PID=70 (parent) idata=101 istack=201

☑ c. PID=71 (child) idata=101 istack=201
PID=70 (parent) idata=200 istack=400

☐ d. PID=71 (child) idata=0 istack=0
PID=70 (parent) idata=200 istack=400

8. **/proc** 1. What useful information can be found? 2. Can any file be changed? why or why not.

The /proc directory in Linux is a virtual filesystem, also called procfs. It is dynamically generated by the kernel and resides in memory, not on the disk. For each running process, a directory is created under /proc using its PID. This contains information like cmdline, cwd, exe, fd, status, stat, etc. The folder also holds system-level information such as /proc/cpuinfo, /proc/meminfo,
and /proc/filesystems, as well as kernel configurations like /proc/sys and /proc/interrupts. Most files in /proc are read-only, reflecting the current state of the system, but some, like those under /proc/sys, are writable for tuning kernel parameters.
macOS does not use a /proc filesystem, as it is based on BSD Unix.

9. Why can't one make a hard link across two file systems?

Inodes are unique to each filesystem and cannot be shared across mounted filesystems (e.g., from /mnt/zfsdrive to /mnt/usb). When a block device is formatted with a specific filesystem (like EXT4, ZFS, or NTFS), it creates its own inode structure, which is incompatible with others. A hard link directly points to a file ' s inode, creating another directory entry for the same data. Since different filesystems have separate inode tables, hard links can't cross filesystem boundaries. Changes to the file through one hard link are reflected in all links, as they all reference the same inode.

(Bonus)10. What are some synchronization techniques when creating processes using fork?

When using fork(), synchronization between processes is essential to prevent race conditions. Common techniques include:
wait()/waitpid(): Ensures the parent waits for the child to finish, preventing zombie processes.
Pipes: Simple, one-way communication between parent and child.
Signals: Useful for event notification but require careful handling.
Shared Memory with Semaphores: Fast and efficient for sharing data, but needs synchronization to avoid conflicts, like counters, mutesx etc.
File Locks: Used to safely access shared files across processes.
Message Queues: Allow structured communication, useful for complex synchronization needs.

11: (Code Submission) (turn in as separate file)

Write a program utilizing fork() to update /proc/sys/kernel/pid_max.

1. **(5pts) Use command line args to input desired value**
2. **(5pts) The child process will read the old value, print it, and update the value (Remember to use a value <= to the old pid_max).**
3. **(5pts) The parent will read and print the new value. (you might need to run your program with sudo to change the value)**
4. **(15pts) You must use proper error handling for all library functions/system calls.**