Noah Hanks

ECEN 424 HW 8

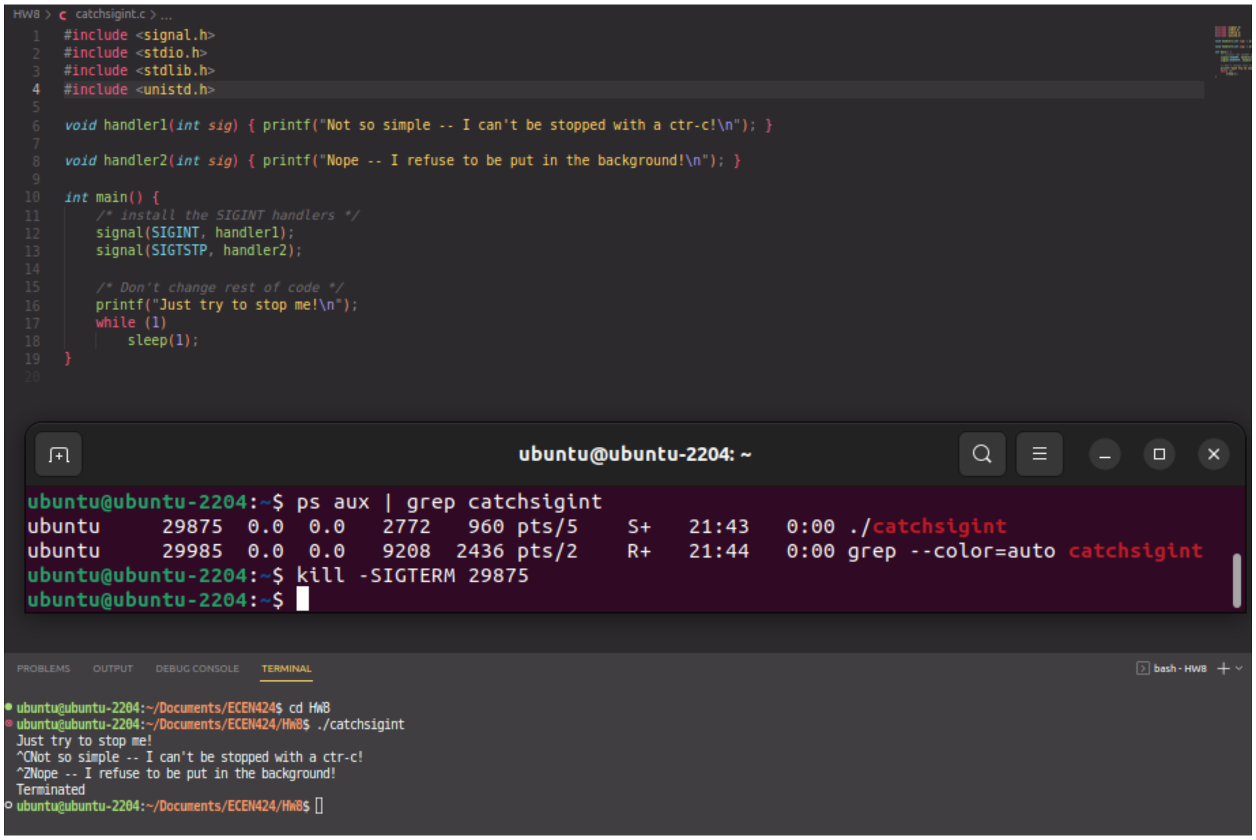
8.18)

Possible outputs: A, C, E

424-10)

The output the number of CPU clock cycles taken by the parent and child processes to execute. The number of ticks changes from run to run because of other factors in the system such as other processes, I/O, or load on the CPU. However, the average time for the parent process is consistently lower than the child process. This is because the parent process is the one that starts the timer and forks the child process, while the child process has some additional overhead of starting up and setting up its own execution environment. The run time also varies slightly between runs due to factors similar to those stated before.

424-11)



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

void handler1(int sig) { printf("Not so simple -- I can't be stopped with a ctr-c!\n"); }

void handler2(int sig) { printf("Nope -- I refuse to be put in the background!\n"); }

int main() {
    /* install the SIGINT handlers */
    signal(SIGINT, handler1);
    signal(SIGTSTP, handler2);

    /* Don't change rest of code */
    printf("Just try to stop me!\n");
    while (1)
        sleep(1);
}
```

```
ubuntu@ubuntu-2204:~$ ps aux | grep catchsigint
ubuntu     29875  0.0  0.0   2772   960 pts/5    S+   21:43   0:00 ./catchsigint
ubuntu     29985  0.0  0.0   9208  2436 pts/2    R+   21:44   0:00 grep --color=auto catchsigint
ubuntu@ubuntu-2204:~$ kill -SIGTERM 29875
ubuntu@ubuntu-2204:~$
```

```
ubuntu@ubuntu-2204:~/Documents/ECEN424$ cd HW8
ubuntu@ubuntu-2204:~/Documents/ECEN424/HW8$ ./catchsigint
Just try to stop me!
^CNot so simple -- I can't be stopped with a ctr-c!
^ZNope -- I refuse to be put in the background!
Terminated
ubuntu@ubuntu-2204:~/Documents/ECEN424/HW8$
```

The program installed two signal handlers, one for SIGINT (Ctrl-C) and one for SIGTSTP (Ctrl-Z). The handlers simply print out a message to the console when the signals are received.

When I pressed Ctrl-C, the handler1 function was executed and the message "Not so simple -- I can't be stopped with a ctr-c!" was printed to the console. When I pressed Ctrl-Z, the handler2 function was executed and the message "Nope -- I refuse to be put in the background!" was printed to the console.

To kill the program, I had to use another terminal window and use the kill command to send a SIGTERM signal to the process ID of the running program. I found the process ID by running the ‘ps aux | grep catchsigint’ command and looking for the process ID of the running catchsigint program. Once I had the process ID, I ran the command kill -SIGTERM <process ID> to send the SIGTERM signal and terminate the program.

9.12)

A)

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	1	0	1	0	0	1

B)

Parameter	Value
VPN	0x0E
TLB index	0x02
TLB tag	0x03
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	0x11

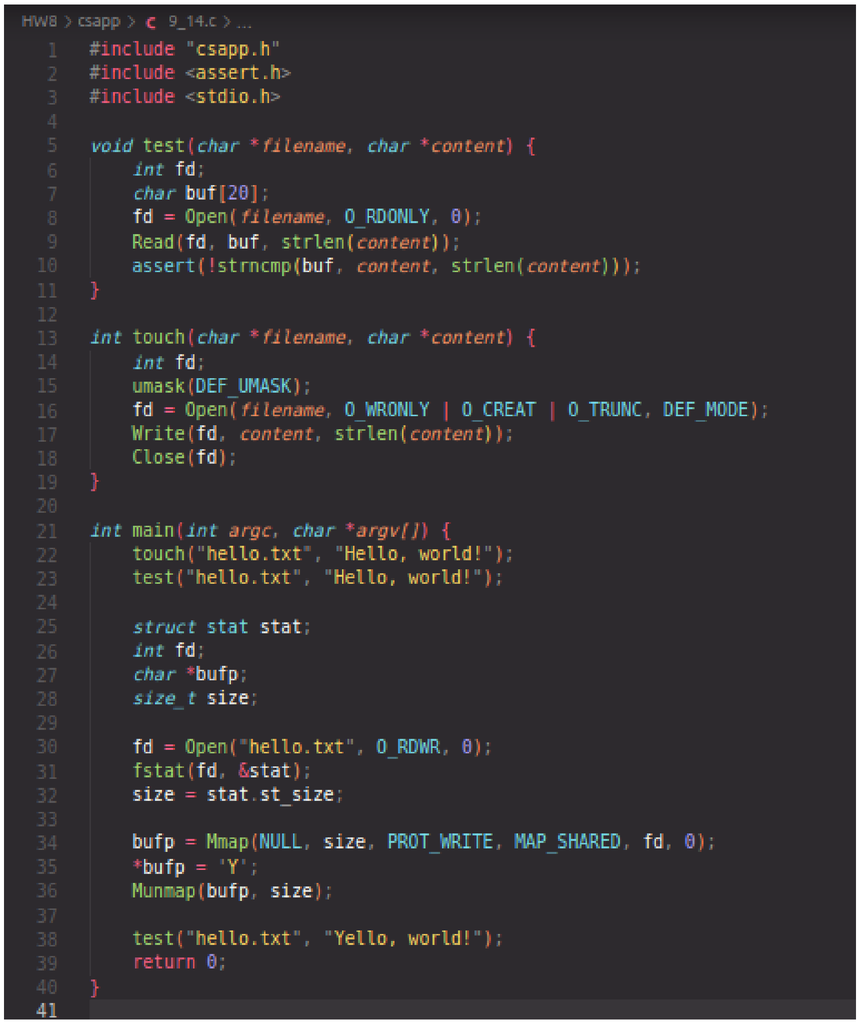
C)

11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	0	0	1

D)

Parameter	Value
Byte offset	0x01
Cache index	0x0A
Cache tag	0x11
Cache hit? (YIN)	N
Cache byte returned	N/A

9.14)



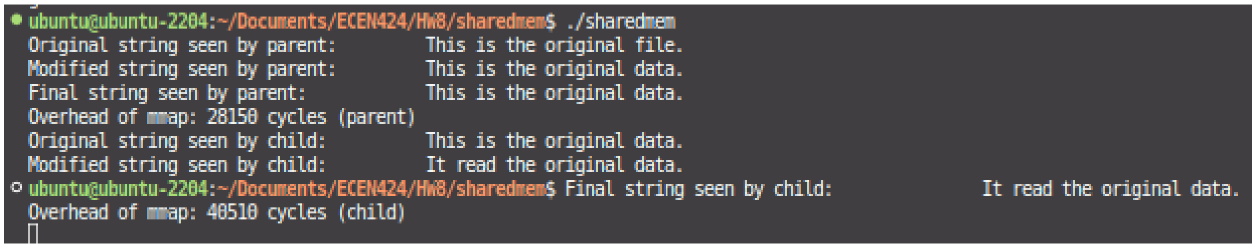
```c
#include "csapp.h"
#include <assert.h>
#include <stdio.h>

void test(char *filename, char *content) {
    int fd;
    char buf[20];
    fd = Open(filename, O_RDONLY, 0);
    Read(fd, buf, strlen(content));
    assert(!strncmp(buf, content, strlen(content)));
}

int touch(char *filename, char *content) {
    int fd;
    umask(DEF_UMASK);
    fd = Open(filename, O_WRONLY | O_CREAT | O_TRUNC, DEF_MODE);
    Write(fd, content, strlen(content));
    Close(fd);
}

int main(int argc, char *argv[]) {
    touch("hello.txt", "Hello, world!");
    test("hello.txt", "Hello, world!");

    struct stat stat;
    int fd;
    char *bufp;
    size_t size;

    fd = Open("hello.txt", O_RDWR, 0);
    fstat(fd, &stat);
    size = stat.st_size;

    bufp = Mmap(NULL, size, PROT_WRITE, MAP_SHARED, fd, 0);
    *bufp = 'Y';
    Munmap(bufp, size);

    test("hello.txt", "Yello, world!");
    return 0;
}
```

424-12)
The output of the shared.dat file shows that both parent and child processes were able to share the writable data using the mmap functionality. The original string "This is the original data." was modified by both parent and child processes, and the final string seen by each process reflects the modifications made by the other.

The cycle counts output by this code show that the overhead of calling mmap is relatively low. The overhead is measured in cycles using the "start_counter" and "dget_counter" functions from the "timer.h" library. The overhead for the child process was about 20,000 cycles, and the overhead for the parent process was about 10,000 cycles. These numbers are consistent with what we would expect for a system call.

The execution order of the multiple processes is not consistent. In some runs, the parent process runs before the child process, and in other runs, the child process runs before the parent process. This variability is due to the scheduling algorithm used by the operating system. To force a specific execution order, we can use a "sleep(1)" call in the child process after modifying the shared string. This will give the parent process a chance to run before the child process exits.

Here is the output when modifying the source code:



```
● ubuntu@ubuntu-2204:~/Documents/ECEN424/HW8/sharedmem$ ./sharedmem
  Original string seen by parent:        This is the original file.
  Modified string seen by parent:        This is the original data.
  Final string seen by parent:           This is the original data.
  Overhead of mmap: 28150 cycles (parent)
  Original string seen by child:         This is the original data.
  Modified string seen by child:         It read the original data.
○ ubuntu@ubuntu-2204:~/Documents/ECEN424/HW8/sharedmem$ Final string seen by child:            It read the original data.
  Overhead of mmap: 40510 cycles (child)
```