BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
First SEMESTER 2018-2019
CS F372 (Operating Systems)
Tutorial Test (Open Book)
Date: 11-11-2018

Time: 50 Min

Max Marks: 40

Q1. Answer in brief: **[2+ 2+4 +4 = 12 Marks]**

a) List out the conditions in which any process of the Unix operating system gets transformed into zombie and an orphan process?

b) Identify the lower level file system algorithm that transforms the file byte offset into a physical disk block?

c) Consider the sequence of code given below. For values of a) fd=1 and b) fd=3, write what happens to the three descriptor entries and the corresponding file table entry with each call to dup2 system call.

```
dup2(fd, 0);
dup2(fd, 1);
dup2(fd, 2);
if (fd > 2)
close(fd);
```

d) List out the key differences between hard link and soft link. In addition, explain the difference between unlink() and close() system calls.

Q2. Consider the following program which includes multiple fork system calls and find out the possible different sequence of outputs from the following outputs: 1) DBPBHBG 2) GDPHBBB 3) DPBBGHB 4) DBPHBGD 5) PDBBHGB **[6 Marks]**

```
int main(void)
{
  pid_t process_id;
  int status;

 if (fork() == 0){
    if (fork() == 0) {
      printf("P");
    }
    else {
      process_id = wait(&status);
      printf("H");
            }
  }
 else {
    if (fork() == 0) {
      printf("G");
      exit(0);
    }
    printf("D");
  }
  printf("B");
  return (0);
}
```

Q3. Consider the following program consisting of multiple fork system calls and estimate the total number of new processes created by this program **[4 Marks]**

```
int main(void)
{
1 pid_t pid = fork();
2 pid = fork();
3 pid = fork();
4 if (pid == 0)
  {
5   fork();
  }
6 fork();
  return 0;
}
```

Q 4. Consider that a logical block on the file system holds 1K bytes and that a block number is addressable by a 32 bit integer, then a block can hold up to 256 block numbers. Assume there are 13 entries in the inode table: 10 direct, 1 indirect, 1 double indirect, 1 triple indirect block. If a process makes a request to access 4,12000 byte offset, then find out which index block contains this byte information. **[4 Marks]**

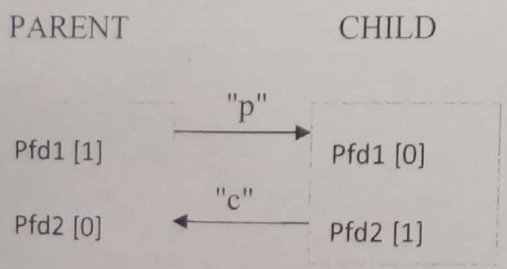
Q 5. Consider the following function, search (char* file, char* dir) that returns 0 if the file is found in the directory and returns 1 otherwise. Complete the following code **[6 Marks]**

```
int search (char* file, char* dir)
{ DIR *dirptr=opendir(dir);
struct dirent *entry = readdir(dirptr);
while (entry != NULL)
{ if ( ------------------- == strlen(file) && (strcmp(-----------------
---) == 0)
     return 0; /* return success */
entry = ----------------;
}
 return 1; /* return failure */ }
```

Q6. Consider that when a pipe is empty, the reader process blocks until writer process write something into pipe. Similarly, when pipe is full, the writer process blocks until reader process reads something from pipe. In the below program, parent and child processes need to communicate with each other according to the following rules:

- Parent sends a message (a character "p") to child.
- Child sends acknowledgment message (a character "c") to parent on receiving "p", otherwise, child waits and does nothing.
- When parent receives "c", it sends the next "p" to child, otherwise, parent waits and does nothing.

Before fork, we create two pipes as shown in the following figure. Parent writes the character "p" across the top pipe when TELL_CHILD is called, child writes "c" across the bottom pipe when TELL_PARENT is called.



Complete TELL_PARENT, TELL_CHILD, WAIT_PARENT, WAIT_CHILD to make parent and child communicate correctly.

[8 Marks]

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

static int Pfd1[2],Pfd2[2];
void TELL_WAIT(void){
    if(pipe(Pfd1)<0 || pipe(Pfd2)<0){
        perror("pipe");
        exit(EXIT_FAILURE);
    }
}

void TELL_PARENT(void){
  /* COMPLETE send parent a message through pipe*/

    printf("Child send message to parent!\n");
}

void WAIT_PARENT(void){
  /* COMPLETE read message sent by parent from pipe*/

    printf("Child receive message from parent!\n");
}

void TELL_CHILD(void){
  /* COMPLETE send child a message through pipe*/

    printf("Parent send message to child!\n");
}

void WAIT_CHILD(void){
  /* COMPLETE read the message sent by child from pipe*/

    printf("Parent receive message from child!\n");
}

int main(int argc, char* argv[]){
    TELL_WAIT();
    pid_t pid;
    pid = fork();
        alarm(10); //set a timer, process will end after 10 seconds.

    if(pid==0){
        while(1){
            sleep(rand()%2+1); TELL_CHILD();WAIT_CHILD();}
    }else{
        while(1){
            sleep(rand()%2+1);
            WAIT_PARENT();
            TELL_PARENT();
        }
      }
    return 0;
}
```