92/100

COSC 350 System Software Midterm #2-1

11/15/2021

Name: Jung An

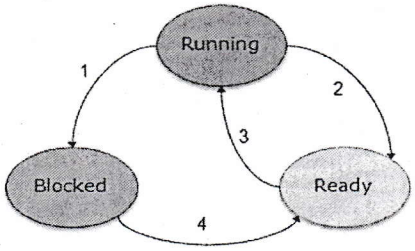
1. (5 pt.)Write a complete C program which creates a zombie process staying in a system forever.

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

int main()
{
  pid_t pid;
  pid = fork();
  if (pid > 0) {
        while (1) {
          sleep(1);
        }
  }
  if(pid == 0){
        exit(0);
  }
  exit(0);
}
```

2. (5 pt.) A process is stayed in one of three states: running, blocked or ready state. Briefly discuss each of states and transaction between states.

- Running state – a process is running
- Ready state – a process is ready to be ran
- Blocked state – a process need another resource and is blocked from running.
- Transaction 1 – while running, it required another resource and goes to blocked state
- Transaction 2 – the process finishes without problem and returns to ready state
- Transaction 3 – recieve signal to run the process and goes to running state
- Transaction 4 – Timed out and goes to ready state



3. (5 pt.) A bash command "env" display all environment variables list. Write a C program which displays all environment variables list. (Do not use **system()** system call with env command)

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

int main (int argc, char **argv, **envp){
    int i;
    for (i=0 ; *(envp[i]) != NULL ; ++) {
        printf("%s\n", *(envp[i]));
    }
}
```

you better be local variable. To assign **envp

4. (5 pt.)
   a. When we write a system program, we should avoid from the race condition. Briefly discuss about race condition.

race condition is when multiple process try to access same resource at the same time and what happen?

   b. Once a child process is created, there is no guarantee which process terminates first. That is depends on the scheduler in the kernel. Each process must keep it's parent process information. What will happen if the parent process terminate before it's child process?

The child process will become an orphan process

-1

systemd its parent become

5. (15 pt.) Write a complete following C program

- Creates a child and grandchild such that three processes try runs forever. Three process must run concurrently.
- The child process keep printing, "child process", parent process keep printing "parent process and grandchild keep printing "grandchild process".
- After printing 10 times, the grandchild sends signal SIGUSR1 to its parent (the child process).
- Once the child process gets SIGUSR1 from grandchild, send SIGUSR2 to its parent and terminate itself.
- When the parent process gets the signal from child, terminate itself.
- Once parent terminated, grandchild recognize somehow and terminate itself.
- **Do not use global variable, do not use wait() or waitpid(), do not use extra signal.**

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

void endPrint(int SIG){
    if (SIG==SIGUSR1){
        kill(getppid(),SIGUSR2);
        exit(0);
    }
    if (SIG==SIGUSR2){
        exit(0);
    }
}

int main(){
    pid_t pid,pid2;
    pid=fork();
    if(pid>0){
        signal(SIGUSR2,endPrint);
        while(1){
            printf("parent process\n");
        }
    }
    if(pid==0){
        pid2=fork();
        if(pid2>0){
            signal(SIGUSR1,endPrint);
            while(1){
                printf("child process\n");
            }
        }
        if(pid2==0){
            int counter=0;
            while(1){
                if(counter==10){
                    kill(getppid(),SIGUSR1);
                }
                printf("grandchild process\n");
parent died →       if(getppid()==1){
                        exit(0);
                    }
            }
        }
    }
    exit(1);
}
```

6. (15 pt.) Write a following C program (**DO NOT USE GLOBAL VARIABLE**)
   A parent process sends the message "I love you" over a pipe to its child process. The child process reads the message and prints it to standard output as "my mom said I love you". Then child process sends the message "I love you too " over a pipe to its parent. The parent process read message and prints it to standard output as "My child said I love you too". Assume that all system calls succeed (no need to error check).

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

#define   R_END 0
#define   W_END 1

int main() {
    pid_t pid;
    int fd[2], fd2[2], nread;
    char buf[BUFSIZ];
    pipe(fd);
    pipe(fd2);
    pid = fork();
    if(pid > 0) {
        close(fd[R_END]);
        close(fd2[W_END]);
        char message[] = "I love you";
        write(fd[W_END], message, strlen(message));
        nread = read(fd2[R_END], buf, BUFSIZ);
        char out[] = "My child said";
        sprinf(buf, "%s %s", out, buf);
        write(0, buf, strlen(buf));
    }
    if(pid == 0) {
        close(fd[W_END]);
        close(fd2[R_END]);
        char message[] = "I love you too";
        nread = read(fd[R_END], buf, BUFSIZ);
        char out[] = "my mom said";
        sprintf(buf, "%s %s", out, buf);
        write(0, buf, strlen(buf));
        write(fd2[W_END], message, strlen(message));
    }
    exit(0);
}
```

COSC 350 System Software Midterm #2-2

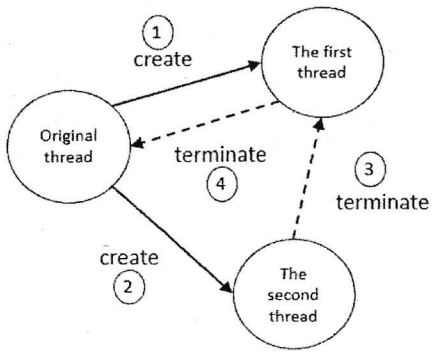
11/17/2021

Name: Jung An

7. (5 pt.) Discuss following concept briefly.

- Race condition – when two or more process access and alter the same resource
then what!?
- Mutual exclusion of critical section – Mutual exclusion allows process to access resources one at a time
- Zombie process – when child process dies while parent process is still running without (x)) wait() or waitpid()
- What are five components of a C program memory layout?
Segment text, unintialized variable, intialized variable, heap, stack

8. (15 pt.) Write following complete C program. **(Do not use signal or global variable!)**
   - Create two threads and each of them will run on the different part of program. And then, **it runs forever** by printing "In the original thread"(sleep one second after). The original thread will be terminated by the first thread. The original thread needs prepare a clean-up handler function Bye1() which will display "END OF PROGRAM" when the original thread is terminated by the first thread.
   - The first thread runs on a function thread1() which **runs forever** by printing "In the first thread" (sleep one second after). The first thread will be terminated by the second thread. The first thread needs prepare a clean-up handler function called Bye() which will call and display "BYE!" when the first thread is terminated by the second thread. Also, the first thread must terminate the original thread. Once original thread is terminated, the second thread will be terminated automatically.
   - The second thread runs on a function thread2() which **runs forever** by printing "In the second thread" (sleep one second after) and count a number increased by one inside a loop. When the number becomes 10, tries to terminate the first thread.



```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

void * Bye(void *arg) {
     printf("BYE!\n");
     pthread_cancel((pthread_t)arg);
     pthread_exit(NULL);
}
void *Bye1() {
    printf("END OF PROGRAM\n");
    pthread_exit(NULL);
}
void *thread1(void *arg){
     pthread_cleanup_push(Bye,(void*)arg);
     while (1) {
        printf("In the first thread\n");
        sleep(1);
        pthread_testcancel();
     }
     pthread_cleanup_pop(0);
}
void *thread2(void *arg){
      int counter=0;
      while(1){
         if(counter==10){
           pthread_cancel((pthread_t)arg);
         }
         printf("In the second thread\n");
         sleep(1);
         counter++;
       }
}
int main()
{  pthread_t threads[2];
   pthread_t self = pthread_self();
   int rc;
   rc = pthread_create(&threads[0], NULL, thread1, (void *) self);
   rc = pthread_create(&threads[1], NULL, thread2, (void *) threads[0]);
   pthread_cleanup_push(Bye1, NULL);
   while (1){
       printf("In the original thread\n");
       sleep(1);
       pthread_testcancel();
   }
   pthread_cleanup_pop(0);
   exit(0);
}
```

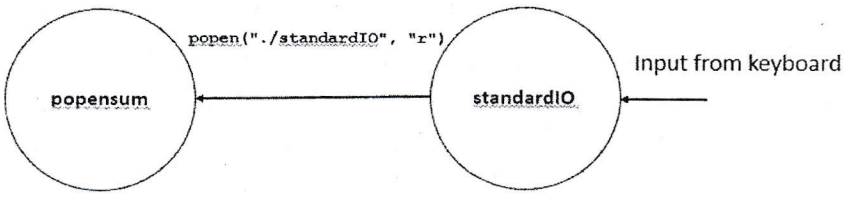
9. Write following two programs
   a. (5pt.) **standardIO.c**
   write a c program which receive c-string (up to 80) from keyboard (standard input) and write on standard output. This program keeps running until Ctr-D (no more data)

```
//stanardIO.c
#include <stdio.h>

int main()
{
    char buf[80];
    int nread;
    while ((nread = read(0, buf, 80)) > 0) {
        write(0, buf, nread);
    }
}
```

   b. (10 pt.) **popensum.c (DO NOT USE GLOBAL VARIABLE)**

   By using **popen**, let a child runs previous program and send data to parent through a pipe. If inputs are two integer values, calculate sum of two numbers and write the result on standard output. If inputs are not two integer values, it must respond as "**invalid inputs: two integers**". This program keeps runs until the child process's (**standardIO** program) termination.



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


int main()
{   char buf[80];
    int num1, num2;
    FILE *ptr = popen("./standardIO", "r");
    while(fgets(buf, 80, ptr)){
        if(sscanf(buf, "%d %d", &num1, &num2) == 2){
            printf("%d\n", num1+num2);
        }
        else{
            printf("invalid inputs: two integer\n");
        }
    }
    exit(0);
}
```

10. (10 pt.) Write a C program which accept an positive integer n as an argument and generates a string of the length specified by n and fills it with random alphabetic characters and display the string on stdout. (use rand()%26 +'a' to create random alphabetic character ASCII code)

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

int main()
{  char *buf;
   char in[BUFSIZ];
   int len;
   read(0, in, BUFSIZ);
   sscanf(in, "%d", &len);
   buf = (char*) malloc(len*sizeof(char));
   int i;
   for(i=0; i<len; ++i) {
      buf[i] = rand()%26 + 'a';
   }
   write(0, buf, len);
   exit(0);
}
```

11. (5 pt.) Write complete C programs which create an orphan process runs forever.

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

int main()
{
  pid_t pid;
  pid = fork();
  if (pid ==0){
        while(1){
          sleep(1);
        }
   }
   if (pid >0){
      exit(0);
   }
  exit(1);
}
```