OS Assignment4

11911419 骆家睿

Q1

- Virtualization: make use of physical resources like CPU and memory, as well as transforms the resources into a more general, powerful and easy-to-use virtual form of itself by providing a few hundred APIs. ----------> Chapter 3,5,9,10
- Concurrency: handle multi-threaded programs and handle different programs running at the same time without triggering off unexpected problems. ----------> Chapter 3,4,5,6,7,8
- Persistence: handle the file system, and protect user data from losing when crashes occur. ----------->Chapter 11,12,13,14,15

Q2

Process A is running -> interrupt by timer -> hardware save Process A's registers onto its kernel stack and switch to kernel mode -> OS saves current register values into memory ,restores register value of process B from memory and changes the stack pointer to Process B's kernel stack -> hardware restores B's register values from its kernel stack and go back to user mode -> Process B runs.

Q3

(1)fork()

- The user calls the fork() system call
- The hardware saves register values and changes to the kernel mode
- The OS takes over the control
    - It creates address space for the new process, and copies the PCB of parent process to this address space
    - It updates the value in the newly created PCB (the PCB of child process), such as PID, running time, etc.
    - It adds the child process to the list of children in parent PCB and adds the pointer to parent process in child PCB.
    - It adds the child process node to the task list.
- The CPU scheduler in OS would decide which process would run next
    - If parent process would be run next:
        - Load data from PCB and the hardware changes to user mode
        - Return the PID of the child process to the parent process and the parent process runs.
    - If child process would be run next(Context switch happens):
        - Load data from PCB and the hardware changes to user mode
        - Return 0 to the child process and child processes runs.

(2)exit()

- The user calls the exit()
- The OS takes over control
  - It would free allocated memory of the process while still maintaining the process ID in the kernel process table
  - The process comes to a zombie state
  - The kernel sends SIGCHLD to the parent process of the exited process
- If the parent of exited process wait for the process, when it receives SIGCHLD, it calls the handler of SIGCHLD signal, which would destroy the child process in kernel space.
- If the parent of exited process does not wait for the process, then nobody would receive SIGCHLD, in which case a zombie process is produced.

Q4.

- system call: process is asking for system services, like opening a file or creating child process. In this case, the OS would serve the system call first and then CPU scheduler would decide which process would run next.
- an exception is raised: synchronous react to abnormal conditions like divided by zero or requests for invalid memory address. In this case, the OS would handle the exception first and CPU scheduler would decide which process would run next.
- interrupt occurs: asynchronous preempt normal execution, like timer interrupt or some notification from hardware. In this case, the OS would handle the interrupt first and CPU scheduler would decide which process would run next.

Q5

After a new process is admitted, the process would be in the ready state. This state would turn to running state when the CPU scheduler dispatches resources to the process. There are several potential states that the running state could turn to. First, the running state could go back to the ready state if an interrupt occurs. Second, if it requests I/O to the hardware or an event wait happens, it would go to the waiting state, which could return ready state when I/O finishes or event completes. Finally, it could terminate if it exits.

Q6

效果展示：

- 运行myshell:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ./myshell
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- Basic Instructions:
  - ps:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ps
    PID TTY          TIME CMD
   1383 pts/0    00:00:00 bash
  13001 pts/0    00:00:00 myshell
  13004 pts/0    00:00:00 ps
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

  - ls:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ls
myshell  myshell.c  test.sh
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- pwd:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ pwd
/home/jiaruiluo11911419/Desktop/OS/ass4
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- 带有一个参数的ls:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ls -a
.  ..  myshell  myshell.c  test.sh
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- cd 命令:
  - cd .. 【通过ls可以看出确实正确地切换了工作目录】

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ cd ..
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS$ ls
ass4  code_lab3.zip  lab3  lab3.zip  lab4  week2ass
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS$
```

  - cd ~ :

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS$ cd ~
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~$
```

  - cd /home :

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:/home$ ls
jiaruiluo11911419
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:/home$
```

- exit退出:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:/home$ exit
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- 其它功能与优化:
  - 允许命令携带多个参数:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ls -a -l
total 36
drwxrwxr-x 2 jiaruiluo11911419 jiaruiluo11911419  4096 3月  14 21:56 .
drwxrwxr-x 6 jiaruiluo11911419 jiaruiluo11911419  4096 3月  14 16:38 ..
-rwxrwxr-x 1 jiaruiluo11911419 jiaruiluo11911419 17856 3月  14 21:43 myshell
-rw-rw-r-- 1 jiaruiluo11911419 jiaruiluo11911419  3517 3月  14 21:45 myshell.c
-rw-rw-r-- 1 jiaruiluo11911419 jiaruiluo11911419    37 3月  14 21:56 test.sh
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

  - 实现which:

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ which python3
/usr/bin/python3
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ which gcc
/usr/bin/gcc
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

  - 能够使用Ctrl+C来终止正在运行的程序: 【以top为例】

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ top

top - 23:22:42 up  6:25,  1 user,  load average: 0.01, 0.04, 0.02
Tasks: 173 total,   1 running, 172 sleeping,   0 stopped,   0 zombie
%Cpu(s):  6.7 us,  6.7 sy,  0.0 ni, 86.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 s
MiB Mem :   3926.2 total,   1823.8 free,    795.3 used,   1307.1 buff/cache
MiB Swap:    448.5 total,    448.5 free,      0.0 used.   2894.9 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    804 jiarui+  20   0  558868  73860  44272 S   6.7   1.8   1:50.66 Xorg
   1042 jiarui+  20   0 3756676 339920 126368 S   6.7   8.5   3:09.46 gnome-+
  13890 jiarui+  20   0   14936   4036   3520 R   6.7   0.1   0:00.01 top
      1 root      20   0  167460  11448   8444 S   0.0   0.3   0:01.63 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.01 kthrea+
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_pa+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworke+
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_per+
      9 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_ta+
     11 root      20   0       0      0      0 S   0.0   0.0   0:02.32 ksofti+
     12 root      20   0       0      0      0 I   0.0   0.0   0:01.23 rcu_sc+
     13 root      rt   0       0      0      0 S   0.0   0.0   0:00.52 migrat+
     14 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_i+
     16 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     17 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kdevtm+
     18 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
     19 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 inet_f+
     20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kauditd
     21 root      20   0       0      0      0 S   0.0   0.0   0:00.01 khungt+
     22 root      20   0       0      0      0 S   0.0   0.0   0:00.00 oom_re+
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

- 不会因为Ctrl+C退出shell：

```
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ^C
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ^C
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$ ^C
jiaruiluo11911419@jiaruiluo11911419-VirtualBox:~/Desktop/OS/ass4$
```

代码展示：

- 输出用户名、主机名、路径
  - 获取用户名：

```
//Get UserName
char *user_name=getenv("USER");
```

  - 获取主机名：

```
//Get machine name
char ff_s[128];
gethostname(ff_s,sizeof(ff_s)-1);
```

  - 获取路径：比对绝对路径和用户根目录，如果绝对路径当中包含用户根目录，则用~代替绝对路径当中的用户根目录部分，否则直接使用绝对路径

```
//Get current path
char *home=getenv("HOME");
char *final_path=get_my_path(home);
```

```c
//Get current path
char* get_my_path(char* tmphome){
        char *pwd=getcwd(NULL,0);// the absolute directory
    char *home=(char*)malloc(strlen(tmphome));
    strcpy(home,tmphome);//copy the home directory
        int counter=0;
        char *home_split=strtok(home,"/");
        while (home_split){ //count how many layers the home directory has, e.g. /home/jiaruiluo11911419 has 2 layers
                counter++;
                home_split=strtok(NULL,"/");
        }
        // the next a few lines are to check whether the absolute directory contains home directory
    if (strlen(pwd)>=strlen(tmphome)){
        char *part_pwd=(char*)malloc(strlen(tmphome));
        int i=0;
        while (*(tmphome+i)!='\0'){
            *(part_pwd+i)=*(pwd+i);
            i++;
        }
        *(part_pwd+i)='\0';
        if (strcmp(part_pwd,tmphome)==0){ //if the absolute directory contains home directory, then ~ should be used
            char *final_path="~";
                char *pwd_split=strtok(pwd,"/");
                for (int i=0;i<counter;i++){
                        pwd_split=strtok(NULL,"/");
                }
                while (pwd_split){
                        char* temp=(char*) malloc(strlen(final_path)+strlen(pwd_split)+strlen("/"));
                        strcpy(temp,final_path);
                        strcat(temp,"/");
                        strcat(temp,pwd_split);
                        final_path=temp;
                        pwd_split=strtok(NULL,"/");
                }
            free(home);
            return final_path;
        }
        free(part_pwd);
    }
    return pwd; //if not contains, then directly output the absolute path
}
```

- 获取命令：

```
    //start reading input
    while (1){
        //read the input command
        char command[1024];
        printf("%s@%s:%s$ ",user_name,ff_s,final_path);
        fflush(stdout);
        ssize_t readFlag=read(0,command,sizeof(command)-1);
        if (readFlag>0){command[readFlag-1]='\0';}
        else{
            printf("Wrong Input\n");
            break;
        }
        char* argv[128];
        argv[0]=command;
        int iter=1;
        char* pointer=command;
        //convert the command into a list of arguments
        while (*pointer){
            if (isspace(*pointer)){
                *pointer='\0';
                pointer++;
                argv[iter++]=pointer;
            }
            else{
                pointer++;
            }
        }
        argv[iter]=NULL;
```

- exit退出：

```
        //exit
        if (strcmp(command,"exit")==0) break;
```

- 实现cd：【使用chdir库函数】

```
        //cd
        if (strcmp(argv[0],"cd")==0){
                                if (iter==1){
                                        printf("[myshell] cd: missing argument\n");
                                }
                                else{
                    if (strcmp(argv[1],"~")==0){
                        argv[1]=getenv("HOME"); //implement cd ~
                    }
                                        chdir(argv[1]);
                                        final_path=get_my_path(home);
                                }
                continue;
            }
```

- 实现which: 【从环境变量PATH所包含的路径中寻找可执行文件】

```
        //which
        if (strcmp(argv[0],"which")==0){
            if (iter==1){
                printf("[myshell] which: missing argument\n");
            }
            else{
                char *tmppaths=getenv("PATH");
                char *paths=(char*)malloc(strlen(tmppaths));
                int ii=0;
                while (*(tmppaths+ii)){
                    *(paths+ii)=*(tmppaths+ii);
                    ii++;
                }
                                char *sp_path=strtok(paths,":");
                                while (*sp_path){
                                        char *temp_comm1=(char*) malloc(strlen(sp_path)+strlen(argv[1])+1);
                                        strcpy(temp_comm1,sp_path);
                                        strcat(temp_comm1,"/");
                                        strcat(temp_comm1,argv[1]);
                                        if (access(temp_comm1,X_OK)==0){
                                                printf("%s\n",temp_comm1);
                                                free(temp_comm1);
                                                break;
                                        }
                                        sp_path=strtok(NULL,":");
                                        free(temp_comm1);
                                }
                free(paths);

            }
            continue;
        }
```

- 对于其它命令，父进程创建子进程，注册SIGINT信号处理方式以解决ctrl+c之后等待子进程；子进程先在当前目录寻找对应可执行文件，如果能找到则直接运行，如果不能，则在环境变量PATH当中寻找对应可执行文件

```
    //other command--create child process
    child=-1;
    child=fork();
    if (child){
        //The parent process, when receive SIGINT, interrupts child, which also prevents itself from being interrupted. The parent process should also wait for the child process.
        signal(SIGINT,(void (*)(int))ctrlc_child);
        wait(NULL);
    }
    else{
        //child process
        char* temp_comm;
        int flag=0;
        //check whether the current path has the executable file, if yes, run it directly
        if (access(argv[0],X_OK)==0){
            temp_comm=argv[0];
                    execvp(temp_comm,argv);
            flag=1;
        }
        //if not, search the environment variable $PATH to find the executable file. If found, run it.
        else{
            char *paths=getenv("PATH");
            char *sp_path=strtok(paths,":");
            while (*sp_path){
                temp_comm=(char*) malloc(strlen(sp_path)+strlen(argv[0])+1);
                strcpy(temp_comm,sp_path);
                strcat(temp_comm,"/");
                strcat(temp_comm,argv[0]);
```

```
                if (access(temp_comm,X_OK)==0){
                                execvp(temp_comm,argv);
                    free(temp_comm);
                    flag=1;
                    break;
                }
                sp_path=strtok(NULL,":");
                free(temp_comm);
            }
        }
        if (flag==0) printf("No command found.\n");
        exit(0);
    }
}
```

  - 处理ctrl C的注册方法：

```
//ctrlc child process
pid_t child=-1;
void ctrlc_child(pid_t child){
    if (child>1) kill(child,2);
}
```