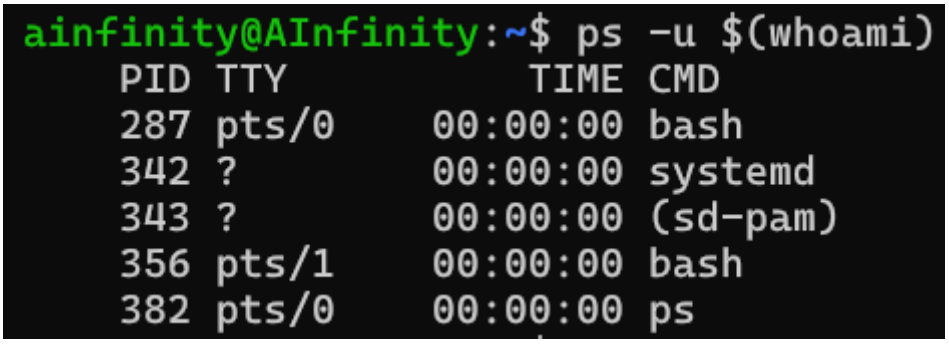
俞楚凡 24302010004 ICS 4.28 lab

这是实验报告。源代码内没有注释，所有代码片段都在实验报告内解释清晰。

2.1：课堂相关命令实践：

关于`ps`命令的使用：

`ps -u $(whoami)`可以指定列出当前用户的进程。`$(whoami)`代表了当前用户。



`ps -aux`显示所有用户的进程。

```
ainfinity@AInfinity:~$ ps -aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.3  0.1  21596 12848 ?        Ss   22:35   0:00 /sbin/init
root           2  0.0  0.0   2616  1444 ?        Sl   22:35   0:00 /init
root           7  0.0  0.0   2616   132 ?        Sl   22:35   0:00 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd
root          52  0.1  0.2  50432 17428 ?        S<s  22:35   0:00 /usr/lib/systemd/systemd-journald
root          93  0.1  0.0  23992  6076 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-udevd
systemd+     148  0.0  0.1  21452 11928 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-resolved
systemd+     149  0.0  0.0  91020  6516 ?        Ssl  22:35   0:00 /usr/lib/systemd/systemd-timesyncd
root         155  0.0  0.0   4236  2728 ?        Ss   22:35   0:00 /usr/sbin/cron -f -P
message+     156  0.0  0.0   9592  5016 ?        Ss   22:35   0:00 @dbus-daemon --system --address=systemd: --nofork --nopidfile -
root         166  0.0  0.1  17976  8184 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-logind
root         169  0.0  0.1 1756096 15860 ?       Ssl  22:35   0:00 /usr/libexec/wsl-pro-service -vv
root         174  0.0  0.0   3160  1156 hvc0     Ss+  22:35   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud - 115200,38400,9
syslog       191  0.0  0.0 222508  5264 ?        Ssl  22:35   0:00 /usr/sbin/rsyslogd -n -iNONE
root         195  0.0  0.0   3116  1080 tty1     Ss+  22:35   0:00 /sbin/agetty -o -p -- \u --noclear - linux
root         201  0.0  0.2 107012 22512 ?        Ssl  22:35   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgr
root         285  0.0  0.0   2624   124 ?        Ss   22:35   0:00 /init
root         286  0.0  0.0   2624   132 ?        S    22:35   0:00 /init
ainfini+     287  0.0  0.0   6204  5348 pts/0    Ss   22:35   0:00 -bash
root         288  0.0  0.0   6696  4716 pts/1    Ss   22:35   0:00 /bin/login -f
ainfini+     342  0.0  0.1  20256 11440 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd --user
ainfini+     343  0.0  0.0  21144  1724 ?        S    22:35   0:00 (sd-pam)
ainfini+     356  0.0  0.0   6072  5116 pts/1    S+   22:35   0:00 -bash
ainfini+     386  0.0  0.0   9584  4684 pts/0    R+   22:37   0:00 ps -aux
ainfinity@AInfinity:~$
```

`ps -auxww`显示进程启动时的参数信息。`-ww`强制显示完整的命令行参数（避免截断）。

```
ainfinity@AInfinity:~$ ps -auxww
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.2  0.1  21596 12848 ?        Ss   22:35   0:00 /sbin/init
root           2  0.0  0.0   2616  1444 ?        Sl   22:35   0:00 /init
root           7  0.0  0.0   2616   132 ?        Sl   22:35   0:00 plan9 --control-socket 6 --log-level 4 --server-fd 7 --pipe-fd
9 --log-truncate
root          52  0.0  0.2  50432 17428 ?        S<s  22:35   0:00 /usr/lib/systemd/systemd-journald
root          93  0.0  0.0  23992  6076 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-udevd
systemd+     148  0.0  0.1  21452 11928 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-resolved
systemd+     149  0.0  0.0  91020  6516 ?        Ssl  22:35   0:00 /usr/lib/systemd/systemd-timesyncd
root         155  0.0  0.0   4236  2728 ?        Ss   22:35   0:00 /usr/sbin/cron -f -P
message+     156  0.0  0.0   9592  5016 ?        Ss   22:35   0:00 @dbus-daemon --system --address=systemd: --nofork --nopidfile -
-systemd-activation --syslog-only
root         166  0.0  0.1  17976  8184 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd-logind
root         169  0.0  0.1 1756096 15860 ?       Ssl  22:35   0:00 /usr/libexec/wsl-pro-service -vv
root         174  0.0  0.0   3160  1156 hvc0     Ss+  22:35   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud - 115200,38400,9
600 vt220
syslog       191  0.0  0.0 222508  5264 ?        Ssl  22:35   0:00 /usr/sbin/rsyslogd -n -iNONE
root         195  0.0  0.0   3116  1080 tty1     Ss+  22:35   0:00 /sbin/agetty -o -p -- \u --noclear - linux
root         201  0.0  0.2 107012 22512 ?        Ssl  22:35   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgr
ade-shutdown --wait-for-signal
root         285  0.0  0.0   2624   124 ?        Ss   22:35   0:00 /init
root         286  0.0  0.0   2624   132 ?        S    22:35   0:00 /init
ainfini+     287  0.0  0.0   6204  5348 pts/0    Ss   22:35   0:00 -bash
root         288  0.0  0.0   6696  4716 pts/1    Ss   22:35   0:00 /bin/login -f
ainfini+     342  0.0  0.1  20256 11440 ?        Ss   22:35   0:00 /usr/lib/systemd/systemd --user
ainfini+     343  0.0  0.0  21144  1724 ?        S    22:35   0:00 (sd-pam)
ainfini+     356  0.0  0.0   6072  5116 pts/1    S+   22:35   0:00 -bash
ainfini+     387  0.0  0.0   9584  4840 pts/0    R+   22:38   0:00 ps -auxww
ainfinity@AInfinity:~$
```

2.2.1

`execsingle.c`一个C程序，内部使用execve调用ls -l命令。

main函数部分：

```c
int main() {
        char *argv[] = {"ls", "-l", NULL};
        char *envp[] = {NULL};
        if (execve("/bin/ls", argv, envp) == -1) {
                printf("execve failed");
                exit(1);
        }
        return 0;
}
```

`execve()`:包含三个参数，第一个指定了要运行的程序位置，第二个为携带的命令行参数，第三个为环境变量。在这里，我们指定了argv的两个参数分别为ls和-l。第一个参数为程序名称，虽然已经在execve中的第一个参数所指定，但这仍然是一种约定俗成的规范。第二个为命令行参数。此处不需要环境变量。

如果返回值为-1，则说明execve失败。此时程序应立即退出。如果成功，则execve永远不会返回，程序将在ls命令执行完之后立刻结束。

```
ainfinity@AInfinity:~/ics/lab9$ ./execsingle
total 80
-rw-r--r-- 1 ainfinity ainfinity     0 Apr 28 21:48 a.out
-rwxr-xr-x 1 ainfinity ainfinity 16216 Apr 28 22:01 exec
-rw-r--r-- 1 ainfinity ainfinity   557 Apr 28 22:12 exec.c
-rwxr-xr-x 1 ainfinity ainfinity 16104 Apr 28 22:06 execsingle
-rw-r--r-- 1 ainfinity ainfinity   292 Apr 28 22:43 execsingle.c
-rwxr-xr-x 1 ainfinity ainfinity 16384 Apr 28 21:55 t
-rw-r--r-- 1 ainfinity ainfinity   692 Apr 28 21:58 t.c
-rwxr-xr-x 1 ainfinity ainfinity 16296 Apr 28 22:22 threadmanager
-rw-r--r-- 1 ainfinity ainfinity   642 Apr 28 22:22 threadmanager.c
ainfinity@AInfinity:~/ics/lab9$
```

2.2.2

`exec.c`一个C程序，内部fork了一个子进程，在子进程内execve，父进程等待子进程结束。

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

int main() {
        pid_t fpid;
        fpid = fork();
        if (fpid == 0) {
                char *argv[] = {"ls", "-l", "/home/ainfinity/ics", NULL};
                char *envp[] = {NULL};
                int status;
                if (execve("/bin/ls", argv, envp) == -1) {
                        printf("execve failed");
                        exit(1);
                }
        }
        else if (fpid < 0) {
                printf("fork failed");
                exit(1);
        }
        else {
                int status;
                wait(&status);
                if (WIFEXITED(status)) {
                        printf("exit normally\n");
                }
                else {
                        printf("exit abnormally\n");
                }
        }
        return 0;
}
```

fpid == 0指示其为子进程，我们在其中执行ls -l。fpid < 0说明fork失败，此时应立即退出。fpid > 0为子进程pid，指示为父进程，则调用wait函数等待。等待到子进程退出时，status为退出的状态码。

```
ainfinity@AInfinity:~/ics/lab9$ ./exec
total 24
drwxr-xr-x 3 ainfinity ainfinity 4096 Apr 21 21:00 attacklab
drwxr-xr-x 4 ainfinity ainfinity 4096 Apr 14 13:39 bomblab
drwxr-xr-x 3 ainfinity ainfinity 4096 Mar 23 10:55 datalab1
drwxr-xr-x 2 ainfinity ainfinity 4096 Mar 31 23:53 lab5
drwxr-xr-x 2 ainfinity ainfinity 4096 Apr  7 15:08 lab6
drwxr-xr-x 2 ainfinity ainfinity 4096 Apr 28 22:45 lab9
exit normally
```

可以看到，子进程进行完毕后， 父进程wait到之后执行后面的status判断，随后输出正常退出的消息。

2.2.3

`threadmanager.c`循环创建了三个子进程，并且逐个使用waitpid函数等待退出，获取返回值并打印。

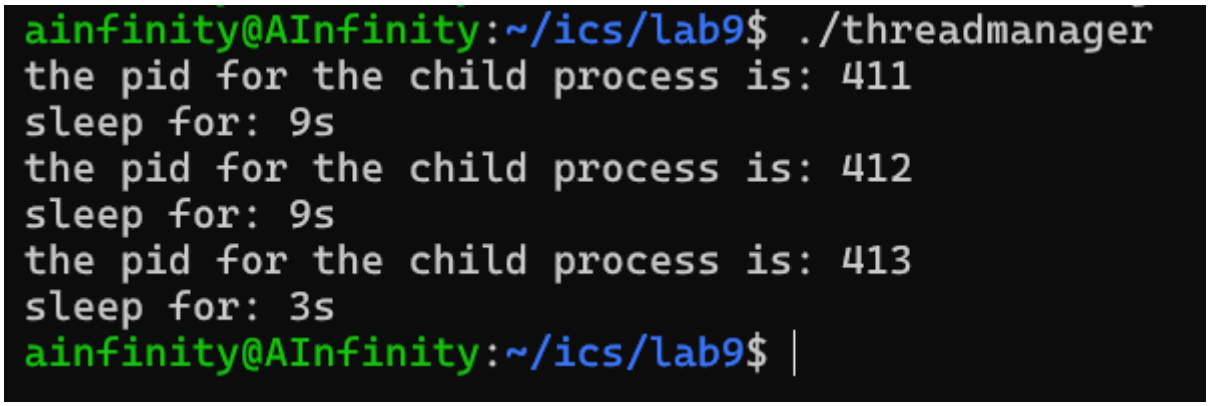
```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/wait.h>

int main(){
        pid_t pidList[4];
        for (int i = 1; i <= 3; ++i) {
                pidList[i] = fork();
                if (pidList[i] == 0) {
                        srand(time(0));
                        unsigned int slt = rand() % 9 + 1;
                        sleep(slt);
                        exit(slt);
                }
                else if (pidList[i] < 0) {
                        printf("fork failed");
                        exit(1);
                }
                else {
                        int status;
                        printf("the pid for the child process is: %d\n", pidList[i]);
                        waitpid(pidList[i], &status, 0);
                        if (WIFEXITED(status)) {
                                printf("sleep for: %ds\n", WEXITSTATUS(status));
                        }
                }
        }
        return 0;
}
```

可以看到，程序使用一个循环fork了三个子进程。我们在每一个循环中，使用waitpid阻塞程序进行直到子进程返回，这样可以确保程序的时序逻辑是同步的。

sleep中的参数为秒数。秒数我们使用srand随机化种子，并且使用rand取随机数。最后返回等待的描述。

waitpid的三个参数分别为子进程的pid，状态码，以及选项。如果不想使用特定选项可以直接赋0.如果status正常，则再利用WEXITSTATUS宏取返回值并输出。



可以看到，前一个子进程退出后再执行下一个子进程的创建和销毁逻辑。