OS systems Dry Part

HW 4

Submitters :
Shaked Or 308026400
Aviad Rozenkof 316592922

Q1

1.

```
int double_wait (int fd1, int fd2){

        //initialize a poolfd array called fds
        struct pollfd fds[2];
        //put the fdnums in the file descriptor field
        (fds[0]).fd = fd1;
        (fds[1]).fd = fd2;
        //put the POLLIN num in the events field
        (fds[0]).events = POLLIN;
        (fds[1]).events = POLLIN;

        return poll (fds, 2, -1);
}
```

2.

This function won't work because SIGUSR1 is still blocked in the beginning of this function.

3.

Imagine the following scenario:
After unblocking SIGUSR1 and before calling poll, SIGUSR1 is signaled.
In this case poll will start after the handler for SIGUSR1 returned. This is problematic because in this case xpoll returns only when SIGUSR1 was signaled and after 1 fd is ready to read (instead of or).

4.

(In this cause we use ppoll thereby making the polling and unmasking atomic)

```
int double_wait_safe(int fd1, int fd2){
        //initialize new sig set with all but SIGUSR1
        sigset_t  newmask;
        sigfillset(&newmask);
        sigdelset(&newmask,SIGUSR1);
        //initialize a poolfd array called fds
        struct pollfd fds_arr[2],          *fds = fds_arr;
        //put the fdnums in the file descriptor field
        (fds_arr[0]).fds = fd1;
        (fds_arr[1]).fds = fd2;
        //put the POLLIN num in the events field
        (fds_arr[0]).events = POLLIN;
        (fds_arr[1]).events = POLLIN;
        //call poll
        return ppoll (fds, 2, -1, &newmask);
}
```

Q2

1.

An adversary user could potentially call RNDCLEARPOOL over and over again, thereby jamming the processes of other users who are waiting for random data.

2.

A.

```
#include <sched.h>
void make_me_FIFO(){
        //make sched_param with MAXPRIO
        struct sched_param param;
        param.sched_priority=0;
        // call setsched
        sched_setscheduler(0,SCHED_FIFO,&param);

}
```

B.

```
#include "/dev/srandom"
#include <sched.h>
void make_me_FIFO(){
        // load the faulty module
        int status;
        status=init_module();
        if(status!=0){
                return -1;
        }
        struct sched_param param;
        param.sched_priority=0;

        //run set_sched until it succeeds
        while(sched_setscheduler(0,SCHED_FIFO,&param)!=0){
                // mix ADDR for chance of getting bit 23 up
                read(NULL,__ADDR__,4,NULL);
        }

}
```

C.

The probability is $\frac{num\ of\ ints\ with\ bit\ 23\ on}{num\ of\ ints}$ which comes out to $\frac{2^{31}}{2^{32}}=\frac{1}{2}$

D.

The expected value is 1/p =2