

## Lezione 17

*Modificare il programma del primo esercizio della Lezione 15 in modo da implementare l'accesso esclusivo al file delle prenotazioni tramite regioni critiche gestite da semafori.*

Il programma riportato di seguito è stato ottenuto da quello della lezione 15, sostituendo le chiamate a `fcntl` con chiamate a `p` e `v` (includendo ovviamente le opportune direttive `include`, `define` e le dichiarazioni delle funzioni ausiliarie):

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/sem.h>
#include <errno.h>

#define SEMPERM 0600
#define MAXLENGTH 3
char buf[MAXLENGTH];

typedef union _semun {
    int val;
    struct semid_ds *buf;
    ushort *array;
} semun;

int p(int semid);
int v(int semid);
int initsem(key_t semkey);
void prenota(int n, key_t semkey);
int acmebook();

main() {
    key_t semkey=0x200;
    int n,n1,n2,fd,pid1,pid2,semid;
    semun ctl_arg;

    while(1) {

        if((fd=open("prenotazioni.txt",O_RDONLY))== -1) {
            perror("Errore in apertura del file delle prenotazioni");
            exit(1);
        }

        /* Lettura dei posti disponibili. */
        if(read(fd,buf,MAXLENGTH)>0) {
            n=atoi(buf);
```

```

        if(n==0) {
            printf("Non ci sono piu' posti disponibili.\n");
            exit(0);
        }
    }
    else
        break;

    close(fd);
    printf("Posti disponibili: %d\n",n);
    /* Viene chiesto il numero di posti da prenotare dall'ufficio A. */
    printf("Numero di posti da riservare dall'ufficio A: ");
    scanf("%d",&n1);
    /* Viene chiesto il numero di posti da prenotare dall'ufficio B. */
    printf("Numero di posti da riservare dall'ufficio B: ");
    scanf("%d",&n2);

    switch(pid1=fork()) {
        case -1:
            perror("Errore nella creazione del primo figlio");
            exit(2);
        case 0:
            prenota(n1,semkey);
        default:

            switch(pid2=fork()) {
                case -1:
                    perror("Errore nella creazione del secondo figlio");
                    exit(3);
                case 0:
                    prenota(n2,semkey);
                default:
                    /* Il padre attende la terminazione dei figli. */
                    waitpid(pid1,NULL,0);
                    waitpid(pid2,NULL,0);
                    semid=initsem(semkey);

                    /* Rimozione del semaforo. */
                    if(semctl(semid,0,IPC_RMID,ctl_arg)==-1) {
                        perror("Errore nella rimozione del semaforo");
                        exit(7);
                    }
            }
        }
    }
}

```

```

}

void prenota(int n, key_t semkey) {
    int i, error_code;

    for(i=0; i<n; i++) {

        /* se acmebook restituisce -1 significa che non vi sono
         * posti disponibili per soddisfare la prenotazione.
         */
        error_code=acmebook(semkey);

        if(error_code== -1) {
            printf("Numero di posti insufficiente.\n");
            exit(error_code);
        }

        if(error_code== -2) {
            printf("Errore nell'inizializzazione del semaforo.\n");
            exit(error_code);
        }

    }

    exit(0);
}

int acmebook(key_t semkey) {
    int error_code=0, fd, n, i, semid;

    if((semid=initsem(semkey))<0)
        return -2;

    /* Apertura in lettura/scrittura del file delle prenotazioni */
    if((fd=open("prenotazioni.txt", O_RDWR))== -1) {
        perror("Errore in apertura del file delle prenotazioni");
        exit(1);
    }

    /* Inizio della regione critica */
    p(semid);

    if(read(fd, buf, MAXLENGTH)>0) {
        n=atoi(buf);

        if(n>0) {
            n--;
            sprintf(buf, "%d", n);

            /* Riposizionamento all'inizio del file prima della scrittura. */

```

```

        if(lseek(fd,0,SEEK_SET)==-1) {
            perror("Errore di riposizionamento nel file delle prenotazioni");
            exit(5);
        }

        /* Il buffer viene 'ripulito' da eventuali caratteri spuri. */
        for(i=strlen(buf);i<MAXLENGTH;i++)
            buf[i]=' ';

        /* Scrittura su disco del nuovo numero di posti disponibili. */
        if(write(fd,buf,MAXLENGTH)==-1) {
            perror("Errore in scrittura nel file delle prenotazioni");
            exit(6);
        }

    }
    else
        error_code=-1;

}

/* Fine della regione critica */
v(semid);

close(fd);
return error_code;
}

int initsem(key_t semkey) {
    int status=0,semid;

    if((semid=semget(semkey,1,SEMPERM | IPC_CREAT | IPC_EXCL))==-1) {
        if(errno==EEXIST)
            semid=semget(semkey,1,0);
    }
    else {
        semun arg;
        arg.val=1;
        status=semctl(semid,0,SETVAL,arg);
    }

    if(semid==-1 || status==-1) {
        perror("initsem fallita");
        return -1;
    }

    return semid;
}

int p(int semid) {

```

```

    struct sembuf p_buf;
    p_buf.sem_num=0;
    p_buf.sem_op=-1;
    p_buf.sem_flg=SEM_UNDO;

    if(semop(semid,&p_buf,1)==-1) {
        perror("p(semid) fallita");
        exit(1);
    }

    return 0;
}

int v(int semid) {
    struct sembuf v_buf;
    v_buf.sem_num=0;
    v_buf.sem_op=1;
    v_buf.sem_flg=SEM_UNDO;

    if(semop(semid,&v_buf,1)==-1) {
        perror("v(semid) fallita");
        exit(1);
    }

    return 0;
}

```