

SISTEMI OPERATIVI A Prova del 4/2/2004

MATR. Cognome. Nome

Corso di Laurea

Username

NOTE

Il presente foglio va immediatamente compilato con le proprie generalità e matricola.

Esso deve essere restituito al termine della prova. In caso di mancata restituzione, la prova dello studente non verrà presa in considerazione per la correzione.

IMPORTANTE

Tutti i file sorgenti prodotti dallo studente per l'esame devono essere memorizzati in un direttorio denominato **soa-040204-x** nella propria home, dove **x** rappresenta il carattere **i** per gli Informatici, **t** per i Telecomunicazionisti, **e** per gli Elettronici.

Soluzioni contenute in altri direttori non verranno prese in considerazione per la correzione.

Prova UNIX-1

Si realizzi in ambiente Unix/C la seguente interazione tra processi:

- il sistema consiste di due processi: un processo P_{padre} che crea un processo P_{figlio} il quale inizialmente attende un segnale SIGUSR1 oppure SIGUSR2 dal processo P_{padre} ;
- il processo P_{padre} , dopo aver atteso un intervallo di durata casuale, invia a P_{figlio} un segnale tra SIGUSR1 e SIGUSR2 per poi inviare, attraverso una pipe p , il numero (1 o 2) del segnale inviato;
- il processo P_{figlio} , ricevuto il segnale e letto il messaggio dalla pipe, visualizza il segnale e il messaggio ricevuti.

Devono essere utilizzate le primitive per la gestione affidabile dei segnali.

Soluzione

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

```
static int segnale_ricevuto;
```

```
void sigusr_handler(int signo)
```

```

{

    segnale_ricevuto = signo ;

}


main()
{
    int pid, piped[2], mesg_segno, segnale_da_inviare;
    struct sigaction act;
    sigset_t sigmask, zeromask;

    /* GESTIONE SEGNALI */

    sigemptyset( &zeromask);

    sigemptyset( &sigmask);
    sigaddset(&sigmask, SIGUSR1);
    sigaddset(&sigmask, SIGUSR2);

    /* Blocco dei segnali SIGUSR1/SIGUSR2 */
    sigprocmask(SIG_BLOCK, &sigmask, NULL);

    act.sa_handler= sigusr_handler; /* Lo stesso gestore per entrambi i segnali */
    sigemptyset( &act.sa_mask);
    act.sa_flags= 0;

    sigaction(SIGUSR1, &act, NULL);
    sigaction(SIGUSR2, &act, NULL);

    /* L'intera politica di gestione dei segnali verra' ereditata del figlio */

    /* CREAZIONE PIPE */
    if(pipe(piped)<0)
    {
        perror("Creazione pipe :");
        exit(-1);
    }

    /* CREAZIONE FIGLIO */

    if((pid=fork())<0)
    {

```

```

        perror("Creazione processo figlio (fork) :");
        exit(-2);
    }
else
    if(pid==0)    /* Processo figlio */
    {
        sigsuspend(&zeromask);

        read(piped[0],&mesg_segnaled, sizeof(int));

        printf("FIGLIO: Ricevuto il segnale %s e il messaggio %d\n",
                (segnale_ricevuto == SIGUSR1)? "SIGUSR1":"SIGUSR2", mesg_segnaled);
        exit(0);
    }
else    /* Processo padre */
    {

        srand(getpid());    /* Per inizializzare il generatore di numeri casuali */
        sleep(rand()%5+1);    /* Attesa di durata casuale tra 1 e 5 secondi */

        segnale_da_inviare = rand()%2 + 1 ; /* 1 o 2 */

        if(segnale_da_inviare == 1)
            kill(pid, SIGUSR1);

        else
            kill(pid, SIGUSR2);

        write(piped[1], &segnale_da_inviare, sizeof(int));

        printf("PADRE: Inviato il segnale %s e il messaggio %d\n",
                (segnale_da_inviare == 1)? "SIGUSR1":"SIGUSR2", segnale_da_inviare);

        wait(NULL);    /* Attesa della terminazione del figlio */

        exit(0);
    }
}

```