PROVA UNIX A

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

1. il sistema consiste di 4 processi: un processo P_M che gestisce l'interazione con l'utente e i tre processi P_A, P_B e P_C ;
2. P_A comunica con P_B e P_C attraverso due pipe dedicate p_b e p_c ;
3. inizialmente, ogni 5 secondi, P_A invia a P_B (attraverso p_b) un numero intero casuale pari e un numero casuale dispari a P_C (attraverso p_c);
4. i processi P_B e P_C memorizzano i dati ricevuti in un proprio file (in formato ASCII) ;
5. se l'utente lo desidera, P_M invia un segnale `SIGUSR1` a P_A che passa ad inviare i numeri casuali pari a P_C (attraverso p_c) e numero casuali dispari a P_B (attraverso p_b); altri segnali `SIGUSR1` fanno ulteriormente commutare il comportamento di P_A;

Utilizzare la gestione affidabile dei segnali.

PROVA UNIX B

Si realizzi in ambiente Unix/C la seguente interazione *client-server* tra processi mediante socket di tipo `Stream`:

1. il sistema consiste di due processi *server* P_{s1} e P_{s2} e di un numero arbitrario di processi clienti P_i ;
2. i processi server P_{s1} e P_{s2} operano alla porta 10001 e forniscono lo stesso servizio di trasferimento file ai processi clienti ;
3. ciascun cliente P_i, ottenuto un nome di file dall'unico argomento di invocazione, richiede tale file contemporaneamente ad entrambi i server e salva localmente il file il cui trasferimento termina per primo ;
4. per ogni nuova richiesta di servizio accettata, i processi P_{si} devono visualizzare l'indirizzo IP del client che si è connesso e creare un nuovo processo figlio.

PROVA UNIX AB

Si realizzi in ambiente Unix/C la seguente interazione *client-server* tra processi mediante socket di tipo `Stream`:

1. il sistema consiste di due processi *server* P_{s1} e P_{s2} e di un numero arbitrario di processi clienti P_i ;
2. i processi server P_{s1} e P_{s2} operano alla porta 10001 e forniscono lo stesso servizio di trasferimento file ai processi clienti;
3. ciascun cliente P_i, ottenuto un nome di file dall'unico argomento di invocazione, richiede tale file contemporaneamente per metà al server P_{s1} e per metà al server P_{s2} ;
4. i processi clienti P_i devono bloccare il segnale `SIGINT`: al termine del salvataggio della copia locale del file ricevuto, in caso di avvenuto invio di un segnale `SIGINT`, devono richiedere all'utente se desidera rimuovere il file ricevuto;
5. per ogni nuova richiesta di servizio accettata, i processi P_{si} devono creare un nuovo processo figlio per la gestione del nuovo cliente;
6. al termine del servizio, ogni processo server figlio scrive nella stessa pipe p_a un messaggio diagnostico con le informazioni di data e ora correnti, IP del cliente connesso e il nome file che è stato servito;
7. i server P_{si} devono terminare la propria attività dopo venti minuti di inattività, dopo aver visualizzato su stdout le informazioni contenute nella pipe p_a.

PROVA TEORIA

1. Si confrontino i concetti di monoprogrammazione e di multiprogrammazione, descrivendo anche la gestione *batch* e il *timesharing*(*Punti 10*).

2. Si dia un esempio di interferenza tra processi concorrenti, la definizione di sezione critica e si illustri lo strumento di sincronizzazione dei semafori, desrivendo la implementazione delle primitive e un esempio di uso (*Punti 20*).