Candidato: Renato Esposito

Matricola: 0124001881

Un sistema è composto di un buffer di N posizioni, 2\*N processi attivi ed un processo coordinatore. Il processo coordinatore estrae uno dopo l'altro in maniera casuale i numeri da 1 a N e ad ogni estrazione i processi competono per aggiudicarsi l'accesso alla posizione corrispondente del buffer, scrivendone il proprio PID. Un processo cha ha scritto il proprio PID nel buffer non partecipa più alla contesa. Quando tutte le posizioni del buffer sono state assegnate, il coordinatore si alterna con l’ultimo processo che ha avuto accesso al buffer nel leggere e stampare, uno alla volta, il contenuto del buffer.

Const N: …

Var:

buffer[N]:int

indice\_globale:int

attiva\_processo: sem(0) // semaforo per attivare un determinato processo

mutex1:sem(1) // semaforo per la mutua esclusione

contatore\_processi: int (0) // un contatore dei processi inizializzato a 0

J: int // indice per il ciclo for di stampa

mutex2:sem(1);

mutex3:sem(1);

mutex4: sem (1);

mutex5:sem(1);

ultimo\_processo: sem(0);

attiva\_coordinatore:sem(0);

SC:sem(1)

Processo coordinatore

Begin

Parbegin

Repeat

While(contatore\_processi != N)

Wait(mutex1);

Indice\_globale=rand(1..N);

Signal(mutex1);

Signal(attiva\_processo)

End while ;

// quando il coordinatore esce dal ciclo while vuol dire che il buffer è stato scritto da N pid per cui è pieno

//attraverso questo ciclo for determino la parità dell’indice , se pari è il turno del processo coordinatore, altrimenti è il turno dell’ultimo processo che è in attesa di essere attivato dal coordinatore

Wait(mutex4)

For j to N

If j == 0

{ LEGGI E STAMPA IL PID DEL BUFFER[J] }

Elseif (j%2==0) // resto=0 INDICE PARI

{ LEGGI E STAMPA IL PID DEL BUFFER[J] }

Else // indice dispari

Signal(ultimo\_processo); // attivo l’ultimo processo

wait(attiva\_coordinatore); // e attendo che mi riattivi

end if;

End for;

Signal (mutex4)

Forever

Parend

End;

Processo Pj… con j= (1…2\*n)

Begin

I : int // indice i per il ciclo for

Parbegin

Repeat

Wait(attiva\_processo);

// effettuo una ricerca al fine di determinare se il processo Pj abbia già il proprio pid nell‘array e lo faccio in sezione critica cosi che nessun altro processo durante questo check possa accedere al controllo if alterano cosi il valore di getpid

Wait (Sc)

For i = 0 to N

If (buffer[i]==getpid()) // ho gia scritto nell’i-esima posizione il mio pid

break(); // esco dal ciclo for

Else

// sono sicuro che in nessuna posizione è presente il mio pid per cui posso inserirlo

Wait(mutex2);

Buffer[indice\_globale]=getpid()

Contatore\_processi++; // incremento la varibile globale che tiene conto di quanti pid sono stati inseriti

Signal(mutex2);

end for;

signal (SC)

wait(mutex3)

//controllo oltre al contatore anche il pid nell’ultima posizione cosicché possa assicurarmi che eventuali prelazioni e nuove schedulazioni di processi Pj non facciano entrare in tale sezione critica un processo che non è ultimo.

if (contatore\_processi == N and getpid()==buffer[indice\_globale]) // vuol dire che il processo Pj è l’ultimo selezionato

wait(ultimo\_processo) // attendo che il coordinatore mi attivi

wait(mutex5)

{ LEGGI E STAMPA IL PID DEL BUFFER[J] }

signal (mutex5)

signal(attiva\_coordinatore); // attivo il coordinatore per ciclare l’indice

end if;

signal(mutex3)

Forever

Parend

End;

Si realizzi un programma in C e Posix sotto Linux che implementi la seguente pipe, secondo la modalità della shell BASH: ls -al | grep root | wc -l

#DEFINE READ 0

#DEFINE WRITE 1

Int main( int argc, char \*argv[]){

Int fd[2];

Pid\_t figlio\_A, figlio\_B;

if( pipe(fd) == -1) {

perror(“Errore pipe”);

exit(EXIT\_FAILURE);

}

figlio\_A=fork();

figlio\_B=fork();

if (figlio\_A==0) // è il figlio che scriverà il comando ls -al

{

Close(fd[READ]);

dup2(fd[WRITE],1 );

close(fd[WRITE]);

execlp(“ls”,”ls”,”-al”,NULL); // esegue il commando ls con estensione -al

} // FINE FIGLIO A

If (figlio\_B==0) // sono nel figlio B che eseguirà il comando greep root

{

// legge da pipe

Close (fd[WRITE]);

Dup2(fd[READ],0);

Close (fd[READ]);

// scriverà l’output nella pipe

Close (fd[READ]); // è già chiusa ma l’ho riscritta per schematizzare le operazioni

Dup2(fd[WRITE],1);

Close (fd[WRITE]);

Execlp (“greep”,”greep”,”root”,NULL);

} // FINE FIGLIO B

If (filgio\_A != 0 && figlio\_B != 0) //PADRE legge da pipe ed esegue il comando wc -l

Close (fd[WRITE]);

Dup2(fd[READ],0);

Close (fd[READ]);

Execlp(“wc”,”wc”,”-l”,NULL);

} // FINE MAIN ()