Qualche esercizio

E. Mumolo

- Chiamate di sistema per la gestione processi in Linux
  - fork, wait, waitpid, exec
  - implementazione di grafi delle precedenze
- Chiamate di sistema per la gestione dei Segnali in Linux
  - kill, sigaction
- Interprocess communications in Linux
  - pipe, fifo, shared memory Posix
- Sincronizzazione tra processi in Linux: semafori Posix
  - sem_init, sem_open, sem_wait, sem_post
  - esercizi di sincronizzazione semaforica di base
- problemi classici di sincronizzazione in Linux
  - Produttore/Consumatore
  - Lettori/scritttori
  - Il negozio del barbiere
- Introduzione ai pthread di Linux
  - creazione thread
  - join thread
  - passaggio del nome della funzione al thread
  - passare argomenti al thread
  - sincronizzare thread con semafori

Gestione file

- Scrivere un codice equivalente al comando cat usando non più di 3 linee di codice nel corpo del main

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define   BUFFSIZE  8192

int main(void)
{
    int    n;
    char   buf[BUFFSIZE];
    while ( (n=read(0, buf, BUFFSIZE)) && write(1, buf, n) ) ;

}
```

Gestione file

- Scrivere un programma che legge un file per righe usando lseek

```
int main(int argc, char **argv){
     FILE*    fd3;
     int      i=0, j, fd1, fd2, n;
     off_t    k, s=0, tab[500][2];
     char     buf[BUFFSIZE], in[3]={"sta"};
     char*    ptr=buf;

     fd1=open(argv[1], O_RDONLY);   fd3=fopen(argv[1],"r");

     while ( fgets(buf,BUFFSIZE,fd3) ){
          n=strlen(buf);
          tab[i][0]=s;
          tab[i][1]=n;
          s+=n;
          i++;
     }

     while(strcmp("q",in)!=0){
          printf("numero riga (q per uscire)  "); scanf("%s",in); j=atoi(in);
          s=tab[j][0];  n=tab[j][1];

          k=lseek(fd1,s,SEEK_SET);

          read(fd1,buf,n);  write(1,buf,n);
     }
     return(0);
}
```

- Chiamate di sistema per la gestione processi in Linux

Cosa viene stampato?

```
#include <stdio.h>
#include <sys/types.h>

int glob = 6;

int main(void)
{
        int    var;

        pid_t pid;

        var = 88;

        fork();

        glob++;

        var++;

        printf("pid = %d"
        " glob = %d"
        " var = %d\n",
        getpid(),
        glob,
        var);

        return(0);

}
```

```
#include <stdio.h>
#include <sys/types.h>

int glob = 6;

int main(void)
{
        int    var;

        pid_t pid;

        var = 88;

        fork();

        glob++;

        var++;

        printf("pid = %d"
        " glob = %d"
        " var = %d\n",
        getpid(),
        glob,
        var);

        return(0);

}
```

Esercizio: cosa viene stampato?

```
#include <sys/types.h>
#include "ourhdr.h"

int glob = 6;

int main(void)
{
    int   var;
    pid_t pid;

    var = 88;
    pid = fork()

    if (pid<0)
          err_sys("errore di fork ");
    else if (pid == 0) { /* processo figlio */
          glob++;
          var++;
    } else{
        sleep(2);/* processo padre */

         printf("pid = %d glob = %d var = %d\n",
          getpid(),

          glob,

          var);

     }

}
```

Esercizio: cosa viene stampato?

```
#include <sys/types.h>
#include "ourhdr.h"

int glob = 6;

int main(void)
{
    int   var;
    pid_t pid;

    var = 88;
    pid = fork()

    if (pid<0)
          err_sys("errore di fork ");
    else if (pid == 0) { /* processo figlio */
          glob++;
          var++;
          printf("pid = %d glob = %d var = %d\n",
          getpid(),
          glob,
          var);
    } else{
        sleep(2);/* processo padre */

         printf("pid = %d glob = %d var = %d\n",
          getpid(),
          glob,
          var);
     }

}
```

```
#include <sys/types.h>
#include "ourhdr.h"

int glob = 6;

int main(void)
{
    int   var;
    pid_t pid;

    var = 88;
    pid = fork()

    if (pid < 0)
          err_sys("errore di fork ");
    else if (pid == 0) { /* processo figlio */
          glob++;
          var++;
         printf("pid = %d glob = %d var = %d\n"
         getpid(),
         glob,
         var);
    } else{
        sleep(2);/* processo padre */

        printf("pid = %d glob = %d var = %d\n",
          getpid(),
          glob,
          var);
     }

}
```

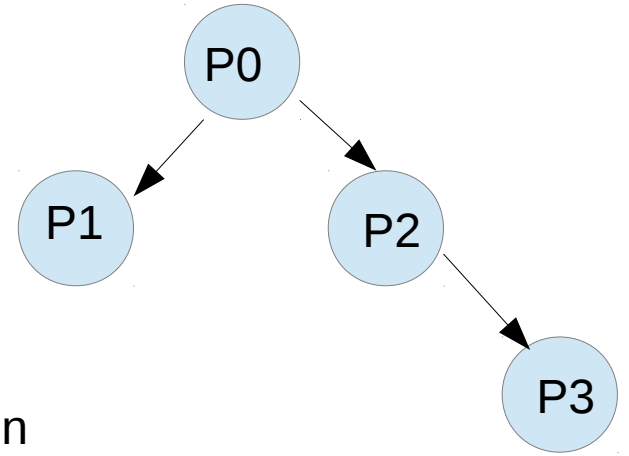
Esercizio: cosa viene stampato?

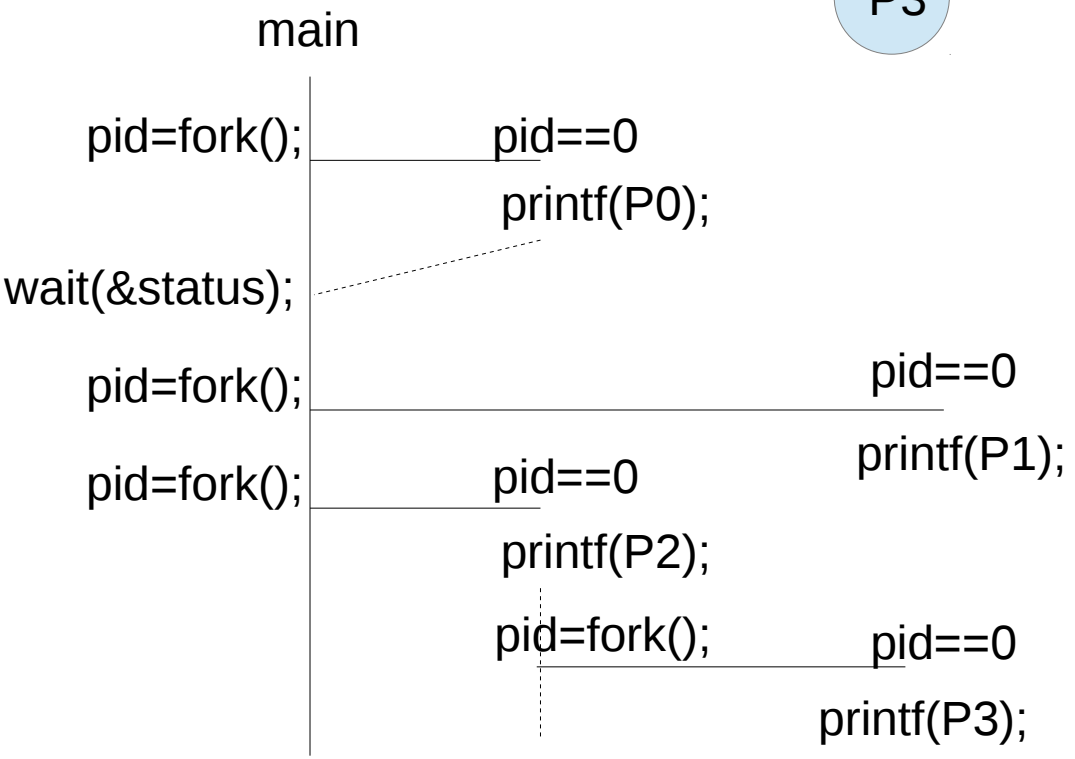
```
#include <sys/types.h>
#include "ourhdr.h"

int glob = 6;

int main(void)
{
    int   var;
    pid_t pid;

    var = 88;
    pid = fork()

    if (pid<0)
          err_sys("errore di fork ");
    else if (pid == 0) { /* processo figlio */
          glob++;
          var++;
    } else{
        sleep(2);/* processo padre */

         printf("pid = %d"
          " glob = %d"

          " var = %d\n",

          getpid(),

          glob,

          var);

     }

}
```

```
#include <sys/types.h>
#include "ourhdr.h"

int glob = 6;

int main(void)
{
    int   var;
    pid_t pid;

    var = 88;
    pid = fork()

    if (pid < 0)
          err_sys("errore di fork ");
    else if (pid == 0) { /* processo figlio */
          glob++;
          var++;
    } else{
        sleep(2);/* processo padre */

        printf("pid = %d"

          " glob = %d"

          " var = %d\n",

          getpid(),

          glob,

          var);

     }

}
```

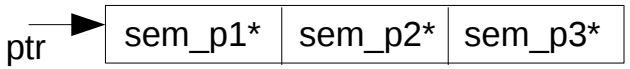
Implementare questo diagramma delle precedenze. P0, P1, P2, P3 sono solo dei nomi scritti con printf. Usare if/else

```
main{
        pid=fork();
        if(pid==0){
                print(P0);
        }
        else{
                wait(&status);
                pid=fork();
                if(pid==0){
                    print("P1");
                }
                else{
                        pid=fork();
                        if(pid==0){
                            print("P2");
                            pid=fork();
                            if(pid==0){
                                    print("P3");
                            }
                        }
                }
        }
}
```

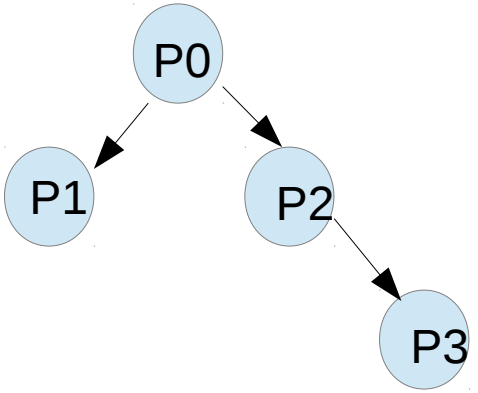




Implementare questo diagramma delle precedenze. P0, P1, P2, P3 sono solo dei nomi scritti con printf. Implementare le precedenze con semafori condivisi



Shared memory

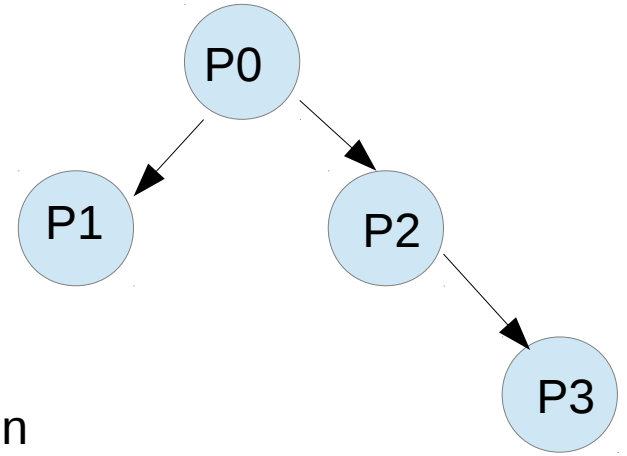


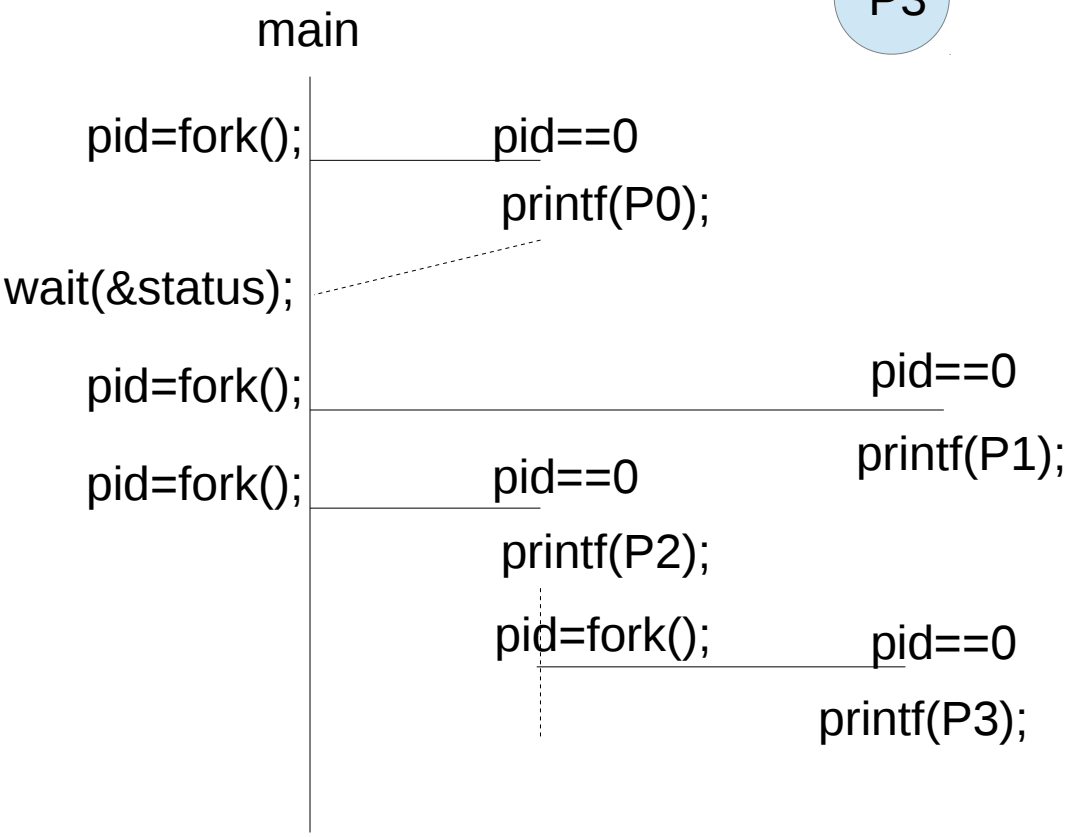

```
int main(int argc, char **argv){
  sem_t* sem_p1;
  sem_t* sem_p2;
  sem_t* sem_p3;
  sem_t* ptr;

  ptr=mmap(0,3*sizeof(sem_t*),PROT_READ|PROT_WRITE,MAP_ANONYMOUS|MAP_SHARED,-1,0);
  sem_p1=ptr; sem_p2=ptr+sizeof(sem_t*); sem_p3=ptr+2*sizeof(sem_t*);
  sem_init(sem_p1,1,0); sem_init(sem_p2,1,0); sem_init(sem_p3,1,0);

  if((pid=fork())==0){
     printf("sono  P0\n");
     sem_post(sem_p1);  sem_post(sem_p2);
     exit(0);
  }
  if((pid=fork())==0){
     sem_wait(sem_p1);  printf("sono P1\n");   exit(0);
  }
  if((pid=fork())==0){
     sem_wait(sem_p2);  printf("sono P2\n");   sem_post(sem_p3); exit(0);
  }
  if((pid=fork())==0){
     sem_wait(sem_p3);  printf("sonoP3\n");
  }

  wait();
  return(0);
}
```

Implementare questo diagramma delle precedenze. P0, P1, P2, P3 sono solo dei nomi scritti con printf. Non usare else

```
main{
    pid=fork();
    if(pid==0){
        print(P0);
        exit(0);
    }
    wait(&status);
    pid=fork();
    if(pid==0){
        print(P1);
        exit(0);
    }
    pid=fork();
    if(pid==0){
            print(P2);
            pid=fork();
            if(pid==0){
                print(P3);
                exit(0);
            }
     }

}
```

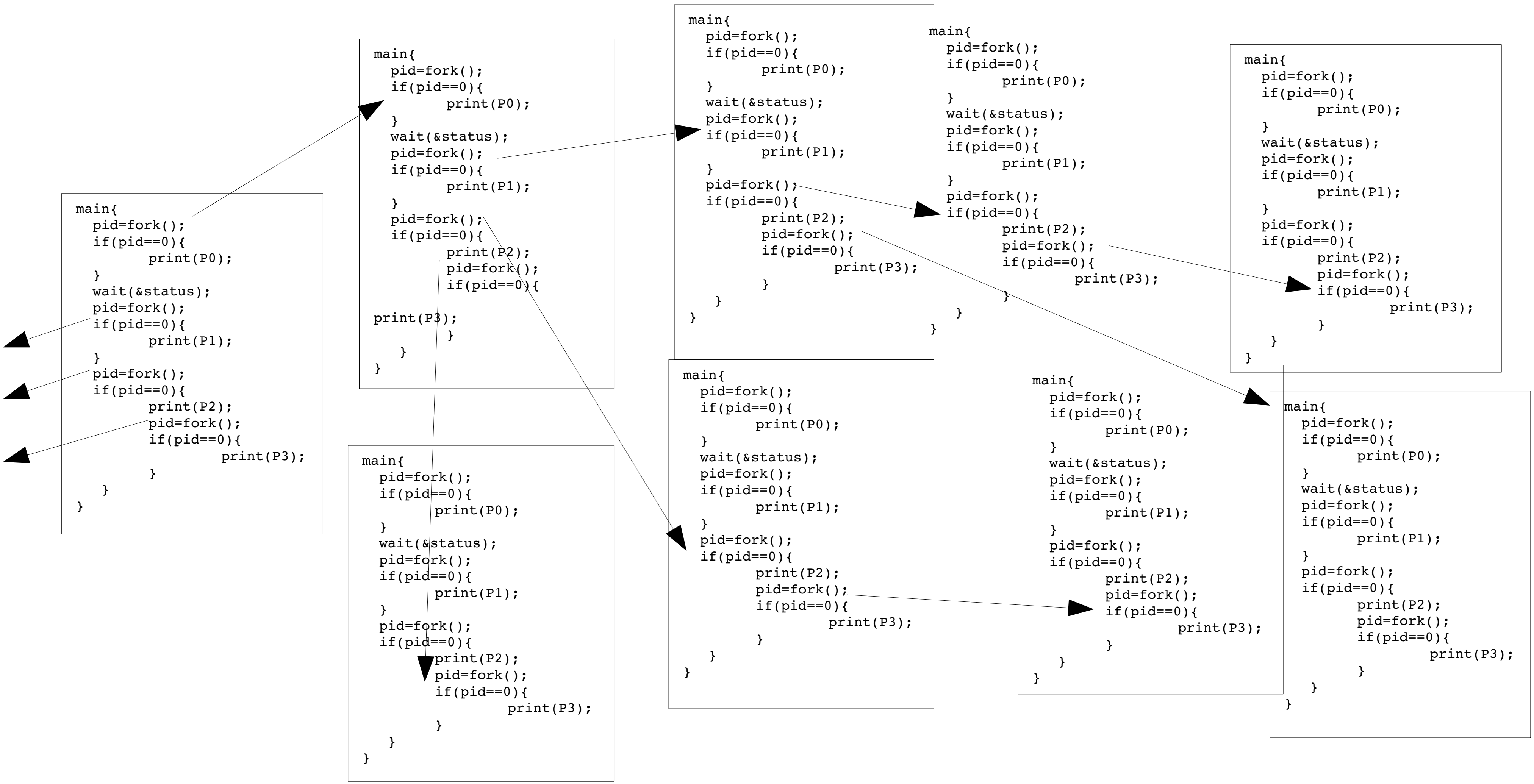
Perchè sono necessari gli exit(0)?





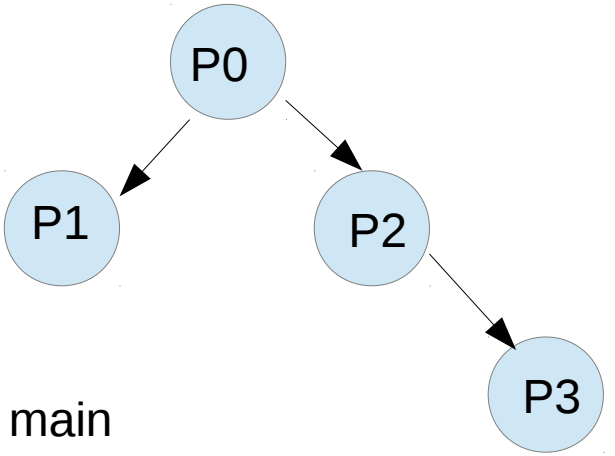
NB. tranne P0 sono tutti processi zombie!

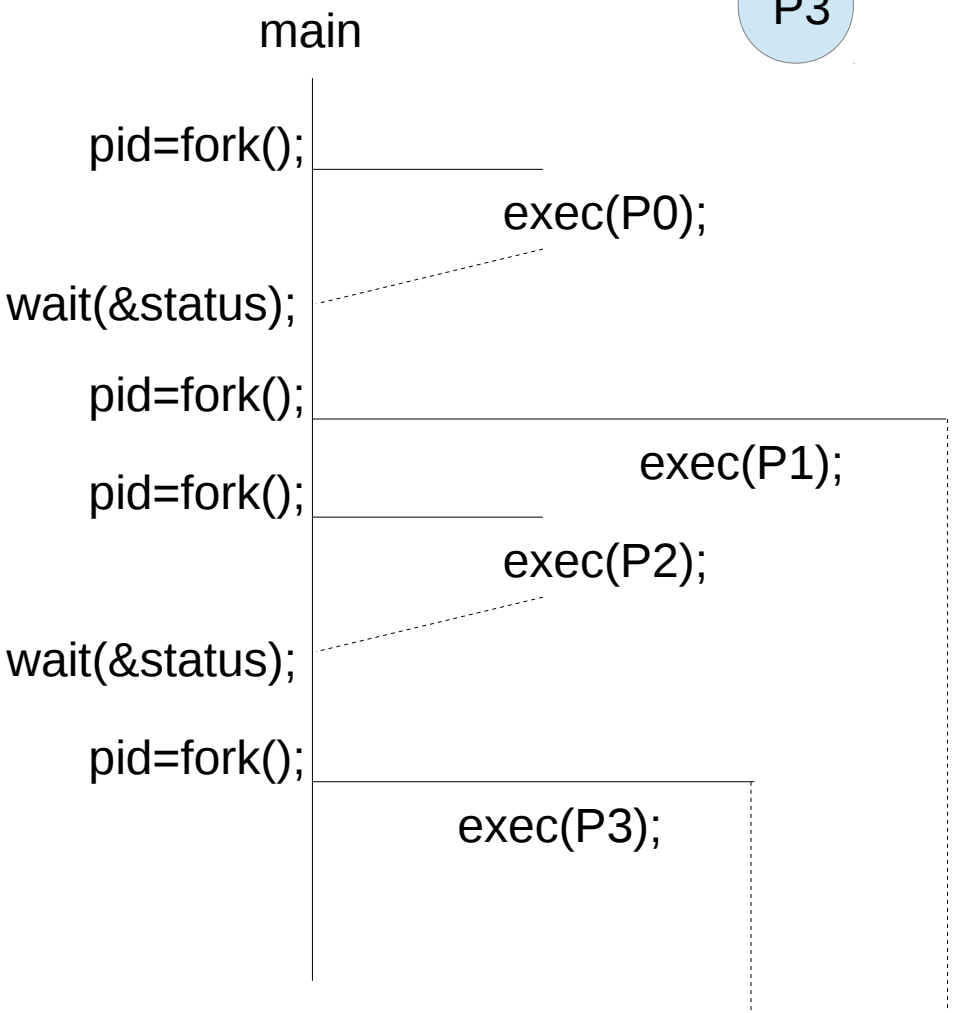
Cosa succede se non metto exit!



Implementare questo diagramma delle precedenze. P0, P1, P2, P3 sono file eseguibili nella directory /bin. Usare if/else

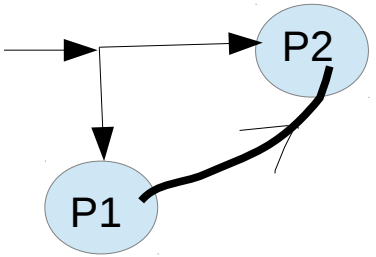
```
main{
    pid=fork();
    if(pid==0){
        exec(P0);
    }
    else{
            wait(&status);
            pid=fork();
            if(pid==0){
                exec(P1);
            }
            else{
                  pid=fork();
                  if(pid==0){
                     exec(P2);
                  }
                  else{
                     wait(&status);
                     pid=fork();
                     if(pid==0){
                        exec(P3);
                     }
                   }
             }
      }
}
```





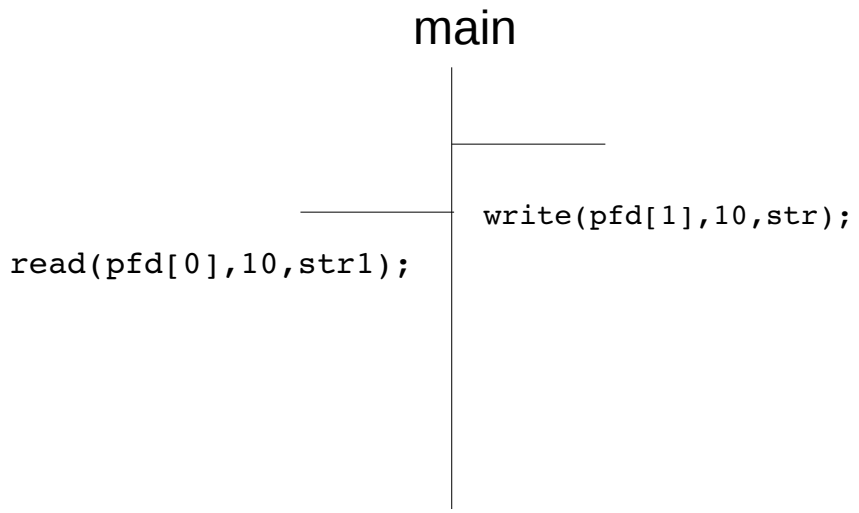
- Interprocess communications in Linux

- Dato questo diagramma delle precedenze



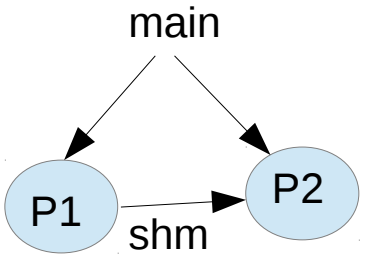
realizzare un canale di comunicazione in direzione P2 → P3 per trasmettere una parola passata dall'utente in linea con l'invocazione del programma. Usare le pipe per realizzare il canale

```
main(int argc, char * argv[])
{
    int status, pfd(2);
    char str[10], str1[10];
    strcpy(str,argv[1]);
    pipe(pfd);

    if(fork()==0){          /*P1*/
           write(pfd[1],10,str);
    }else{
        if(fork()==0){      /*P2*/
            read(pfd[0],10,str1);
            printf("%s",str1);
        }
      }
}
```

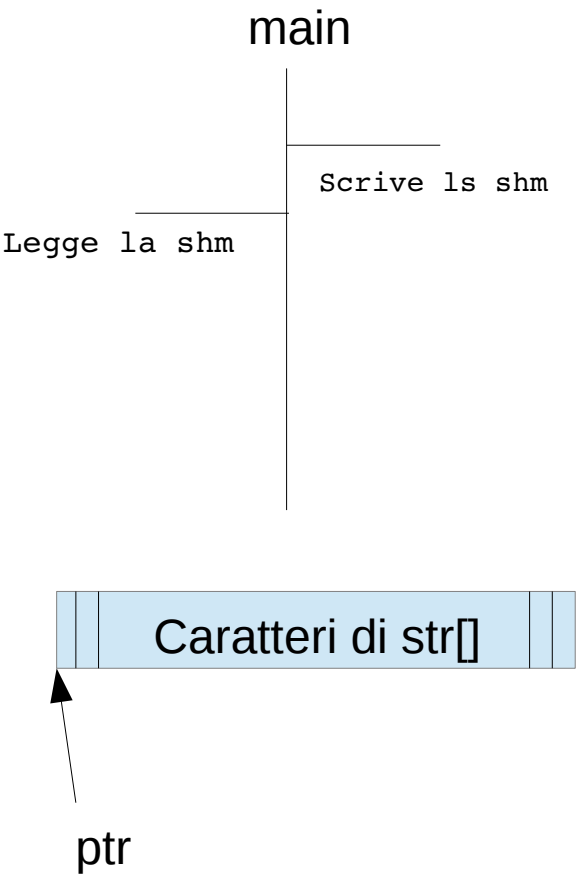


- Dato questo diagramma delle precedenze

realizzare un canale di comunicazione in direzione P1 → P2 per trasmettere una parola passata dall'utente in linea con l'invocazione del programma e modificata da P1. Usare SHM e semafori per realizzare il canale



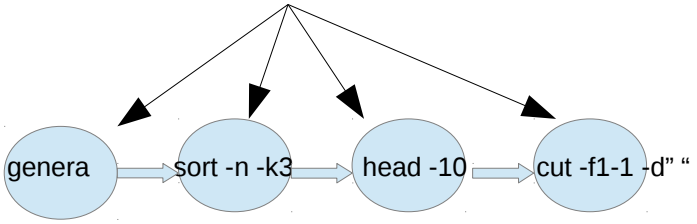

```
main(int argc, char * argv[]){
     int pfd(2); char str[10], str1[10];
     short len;
     char* ptr;
     sem_t s;

     strcpy(str,argv[1]);
     len=strlen(str);
     ptr=mmap(0, len, 0);  /* len byte di shm */
     sem_init(&s,1,0);   /*semaforo senza nome*/

     if(fork()==0){            /*P1*/
              str=modifica(str);
              while (ptr++ = str++);
              sem_post(&s);
     }else{
          if(fork()==0){       /*P2*/
              sem_wait(&s);
              for(i=0;i<len;i++)
                  str1[i]=ptr[i];
              Printf("%s", str1);
          }
      }
}
```



- Sia dato un programma (chiamato genera) che genera un elenco di nomi associati a numeri telefonici, col formato “cognome nome numero”, e che lo scrive su stdout.
  Scrivere un programma che scrive i primi 20 cognomi associati ai numeri telefonici ordinati in ordine crescente.

```
main(){
     int pfd1(2), pfd2(2), pfd3(2);

     pipe(pfd1), pipe(pfd2); pipe(pfd3);
     if(fork()==0){
          dup2(pfd1[1],1); exec(“genera”);
     }else
     {
          if(fork()==0)
          {
               dup2(pfd1[0],0); dup2(pfd2[1],1);
               exec(“sort,”-n”,”-k3”);
          }else
            {
               if(fork()==0){
                    dup2(pfd2[0],0); dup3(pfd2[1],1);
                    exec(“head,”-10”);
               }else
               {
                       if(fork()==0){
                          dup2(pfd3[0],0);
                          exec(“cut,”-f1-1”,”-d” “”);
                       }
                }
             }
     }
}
```

Una possibile architettura del programma:



Processi concorrenti

- Sincronizzazione tra processi in Linux:
  semafori Posix

Si consideri un programma multiprocesso costituito dai processi scrivi1 e scrivi2, dove la variabile “i” è condivisa

```
main()
{
       while(true){
         i=1;
         printf(“Scrivi1:%d”,i);
       }
}
```

```
main()
{
       while(true){
         i=2;
         printf(“Scrivi2:%d”,i);
       }
}
```

Lo scopo e’ semplicemente quello di scrivere Scrivi1:1 e Scrivi2:2 rispettivamente, ma ovviamente ci sono degli interleaving per cui si possono avere delle scritture (errate) tipo: Scrivi1:2 Scrivi2:1 …

Risolvere questo problema usando i semafori. Scrivere lo pseudo-codice di un main che chiama i due processi. Il main crea semafori e memoria condivisa.

```
main(){
   mtx=sem_open("mutex",O_CREAT);

   fd = shm_open("mem",  O_CREAT );/* creo la shmem */
   size=sizeof(int); ftruncate(fd,size);
   ptr = mmap(NULL ,size,fd);
   if(fork()==0) exec(scrivi1);
   if(fork()==0) exec(scrivi2);
}
```

Processo scrivi1

```
void main()
{
       mtx=sem_open("mutex",O_EXCL);
       fd = shm_open("mem",O_EXCL);
       ptr = mmap(NULL ,size,fd);
       while(true){
         sem_wait(mtx);
         *ptr=1;
         printf(“Scrivi1:%d”,*ptr);
         sem_post(mtx);
       }
}
```

Processo scrivi2

```
void main()
{
       mtx=sem_open("mutex",O_EXCL);
       fd = shm_open("mem",O_EXCL);
       ptr = mmap(NULL ,size,fd);
       while(true){
         sem_wait(mtx);
         *ptr=2;
         printf(“Scrivi2:%d”,*ptr);
         sem_post(mtx);
       }
}
```

Si scriva lo pseudocodice di tre processi concorrenti che risolvono il seguente problema:
Si vogliono sommare gli elementi di un array condiviso 'buf' di 20 float, cioè fare l'operazione buf[0]+buf[1]+…+buf[19]. L'operazione viene fatta in modo tale che il primo processo somma gli elementi pari dell'array, mentre il secondo somma gli elementi dispari. Il terzo processo somma i risultati dei primi due. Si proponga una soluzione di mutua esclusione.

```
main(){
   s1=sem_open("s1"); s2=sem_open("s2");

   fd = shm_open("mem",  O_CREAT ); /* creo la shmem */
   size=22*sizeof(float); ftruncate(fd,size);
   ptr = mmap(NULL ,size,fd);
   inizializza_shm(ptr);
   if(fork()==0) exec(sum1);
   if(fork()==0) exec(sum2);
   if(fork()==0) exec(add);
}
```

Processo sum1

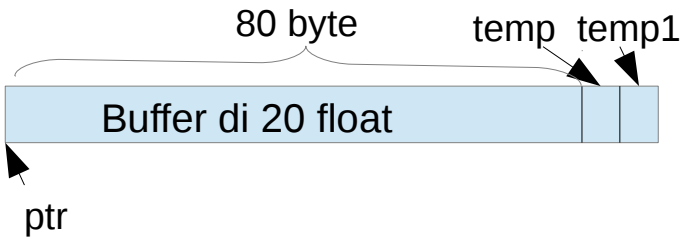
```
void main(){
       s1=sem_open("s1");
       fd = shm_open("mem",O_EXCL);
       ptr = mmap(NULL ,size,fd);
       for(i=0;i<20;i+2)
            temp+=ptr[i];
       *(ptr+20*sizeof(float))=temp;
       sem_post(s1);
}
```

Processo sum2

```
void main(){
       s2=sem_open("s2");
       fd p1=== shm_open("mem",O_EXCL);
       ptr = mmap(NULL ,size,fd);
       for(i=1;i<20; i+2)
            temp+=ptr[i];
       *(ptr+21*sizeof(float))=temp;
       sem_post(s2);
}
```

Processo add

```
void main(){
       s2=sem_open("s1");s2=sem_open("s2");
       fd = shm_open("mem",O_EXCL);
       ptr = mmap(NULL ,size,fd);
       sem_wait(s1); sem_wait(s2);
       temp1=(float)*(ptr+20*sizeof(float));
       temp2=(float)*(ptr+21*sizeof(float));
       sum=temp1+temp2;
}
```



Considerare il seguente problema:. Due processi condividono un array di interi ‘buf’ e l’indice all’array ‘i’ . Si vogliono scrivere tutti gli elementi dell’array senza duplicazioni o mancanze.

```
processo1()
{
     for(i=0;i<N;i++)){
          printf(buf[i];

     }
}
```

```
processo2()
{
     for(i=0;i<N;i++){
          printf(buf[i]);

     }
}
```

Rispondere alle seguenti domande:

a) Ci possono essere problemi nella esecuzione concorrente dei due processi?
b) Se si, specificare di quale problema si tratta

Naturalmente questi processi hanno molti problemi di mutua esclusione.
Se ho un context switch dopo un printf, posso duplicare la stampa e incrementare due volte l'indice.

Soluzione: proteggere con un semaforo binario le righe

```
{
     for()printf();
}
```

- Problemi classici di sincronizzazione in Linux

Una piccola stazione di lavaggio auto può accettare solo auto di lunghezza limitata. Scrivere due thread, Auto e Lavaggio, sapendo che la lunghezza delle auto può essere misurata solo dopo che l'auto sia arrivata. Scrivere i thread in modo che non ci siano corse critiche.

```
Processo  Auto
{
      if(lunghezza<soglia){
            up(auto);
            down(mutex);
            n_auto++;
            up(mutex);
            down(lavaggio);
            Aspetta_fine_lavaggio();
      }
}
```

```
Processo  Lavaggio
{
      while(1){
            down(auto);
            up(lavaggio);
            LavaggioAuto();
            down(mutex);
            n_auto--;
            up(mutex);
      }
}
```

3 semafori: auto, inizializz. a zero, lavaggio, inizializz. al numero di stazioni lavaggio, mutex iniz. a zero

Implementare il problema dei lettori scrittori facendo in modo di scrivere quando esegue il primo e l'ultimo lettore

```
lettore1(){
  while(true){
    down(mutex);
    nrlettori++;
    if(nrlettori==1)
         printf(“primo lettore”);
    up(mutex);
    leggi_archivio();
    down(mutex);
    nrlettori--;
    if(nrlettori==0)
         printf(“ultimolettore”);
    up(mutex);
  }
}
```

```
lettore2(){
  while(true){
    down(mutex);
    nrlettori++;
    if(nrlettori==1)
         printf(“primolettore”);
    up(mutex);
    leggi_archivio();
    down(mutex);
    nrlettori--;
    if(nrlettori==0)
         printf(“ultimo lettore”);
    up(mutex);
  }
}
```

- Introduzione ai pthread di Linux

Scrivere un programma che scrive due pthread che ricevono come argomento l'indirizzo di un array di 100 float, trovano il min e max rispettivamente. Il main normalizza l'array.

```
#include ...
float arr[100]={1.0,1.3,0.5,,,,,,,,,,}
float min=1e6, max=0; /*variabili globali*/


min_funz(void *arg){
          int i; float* buf=(float*)arg;
          for( i=0; i<100; i++) {
               if(buf[i]<min) min=buf[i];
          }
}
max_funz(void *arg){
          int i; float* buf=(float*)arg;
          for( i=0; i<100; i++) {
               if(buf[i]>max)max=buf[i];
          }
}
int main(void) {
     int i;
     pthread_t thr1, thr2;
     pthread_create( &thr1, NULL, (void*)min_funz, (void*)arr );
     pthread_create( &thr2, NULL, (void*)min_funz, (void*)arr );

     pthread_join ( &thr1, NULL );
     pthread_join ( &thr2, NULL );

     for(i=0;i<100;i++)
          buf[i]=(buf[i]-min)/(max-min);
}
```

Scrivere 8 pthread che ricevono come argomento di input una stringa di otto caratteri minuscoli e li stampa maiuscoli sulla console

```
#include ...
int i=0;

funz(void *arg){
          int i; char* str=(char*)arg;
          sem_wait(m);
          i=i+1;
          str[i]+=97;
          sem_post(m);
}

int main(void) {
     char str[8]=”abcdefgh”; int n;
     pthread_t* thr[8];
     sem_t m;

     sem_init(&m,1,1);
     for(n=0;n<8;n++)
         pthread_create(thr[n], NULL, (void*)funz, (void*)str );

     for(n=0;n<8;n++)
         pthread_join(thr[n], NULL);

     printf(“%s\n”,str);
}
```