

Università degli Studi di Genova

Programmazione Concorrente Algoritmi Distribuiti

Lorenzo Vaccarecci

Indice

	Processi e Thread			
	1.1	Processi	2	
		1.1.1 Unix	2	
		1.1.2 Pipe	3	
	1.2	Thread	٠	
		1.2.1 POSIX	9	

Capitolo 1

Processi e Thread

1.1 Processi

Un processo è un programma in corso di esecuzione. Ogni processo necessita di uno spazio di indirizzamento dedicato (stack) e di input e output.

Più processi possono essere eseguiti sulla stessa macchina in un modo quasi simultaneo, un processo non ha accesso allo spazio di indirizzamento degli altri e permette di avere un'illusione di parallelismo.

1.1.1 Unix

Creazione di un processo

```
pid_t fork(void);
```

Ritorna 0 al figlio e il PID del figlio al padre.

fork() crea un processo figlio che è una copia esatta del padre, con lo stesso codice, dati e spazio di indirizzamento. Le variabili non sono condivise.

```
#include <stdio.h>
                           #include <stdlib.h>
                           #include <string.h>
                           #include <unistd.h>
                           #include <sys/wait.h>
                         int main() {
                                       int id=getpid();
                                        printf("Sono il %d\n",id);
                                        printf("Faccio un fork\n");
                                          int idfork=fork();
                                          if(idfork==0){
                                                    printf("Sono %d figlio di %d\n", idfiglio, idpadre); | figlio del figlio di %d\n", idfiglio, idpadre); | figlio del figlio di %d\n", idfiglio, idpadre); | figlio del figlio del
   Siz
lsb
pedv's
 che
                                                      int stato;
del
                                                       waitpid(idfork, &stato, 0);
                                           return 0;
```

Esecuzione di un comando

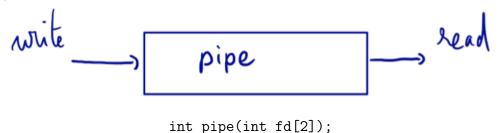
```
int execvp(const char *file, char *const argv[]);
```

Dove file è il nomde del comando che vogliamo eseguire e l'array argv contiene:

- Il nome del comando nel primo campo
- Gli argomenti del comando
- NULL per indicare la fine dell'array in posizione finale obbligatoria

1.1.2 Pipe

Servono per la comunicazione fra processi. Le pipes sono dei canali unidirezionali considerati come dei descrittori di file.



Dove fd[0] è il descrittore di lettura e fd[1] è il descrittore di scrittura. Dopo il forkk, entrambi i processi possono scrivere e leggere ma perchè funzioni bisogna chiudere il descrittore che non si usa.

1.2 Thread

Un thread è un filo d'esecuzione dentro un programma, è eseguito da un processo e possono esserci più thread in un processo (processo multi-thread). Ogni thread è diverso e ha come attributi:

- Un puntatore di esecuzione o PC (Program Counter)
- Uno stack

1.2.1 **POSIX**

#include <pthread.h>

Bisogna anche dire al momento della compilazione che si desidera usare questa libreria con il flag -pthread o -lpthread.

Creazione di un thread

Per condividere le variabili si usa la keyword volatile.

Locks

L'accesso ai dati condivisi deve essere protetto:

- Un thread che desidera accedere a un dato condiviso richiede il lock
- Se il lock è libero, prosegue altrimento rimaane bloccato finchè il lock non è libero
- Quando ha finito, rilascia il lock

Questi lock sono condivisi tra i threads.

E' buona pratica far liberare il lock dal thread che lo ha preso.

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
  int pthread_mutex_lock(pthread_mutex_t *mutex)
  int pthread_mutex_unlock(pthread_mutex_t *mutex)
```

Queste due funzioni ritornano 0 se tutto è andato a buon fine.