Università di Roma Tor Vergata
Corso di Laurea triennale in Informatica

Sistemi operativi e reti

A.A. 2018-2019

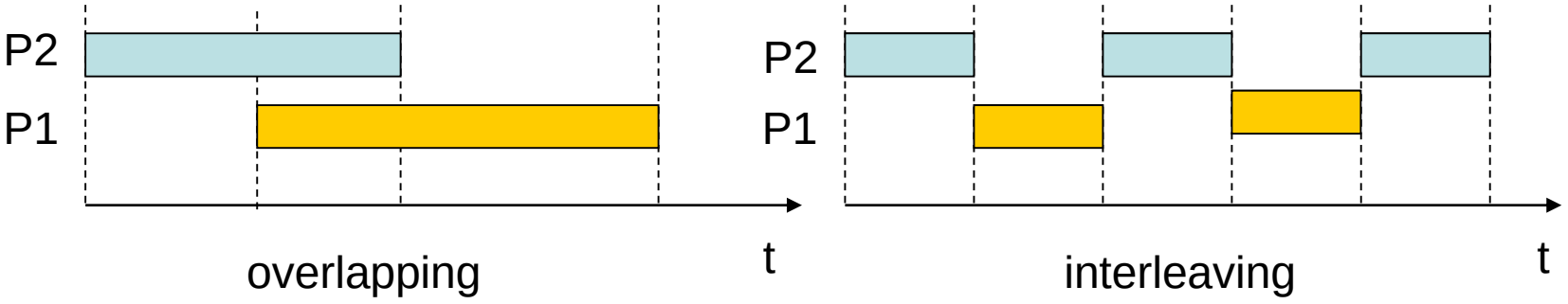
Pietro Frasca

Lezione 6

Giovedì 18-10-2018

Comunicazione tra processi

- I processi in esecuzione in un sistema, sia concorrentemente o in parallelo, possono essere processi *indipendenti* o processi *cooperanti*.
- Il termine ***processi concorrenti*** indica un insieme di processi la cui esecuzione si sovrappone nel tempo. La figura mostra il caso in cui ogni processo è eseguito su un diverso processore *(**overlapping**)* e il caso in cui più processi condividono la stessa CPU *(**interleaving**).*



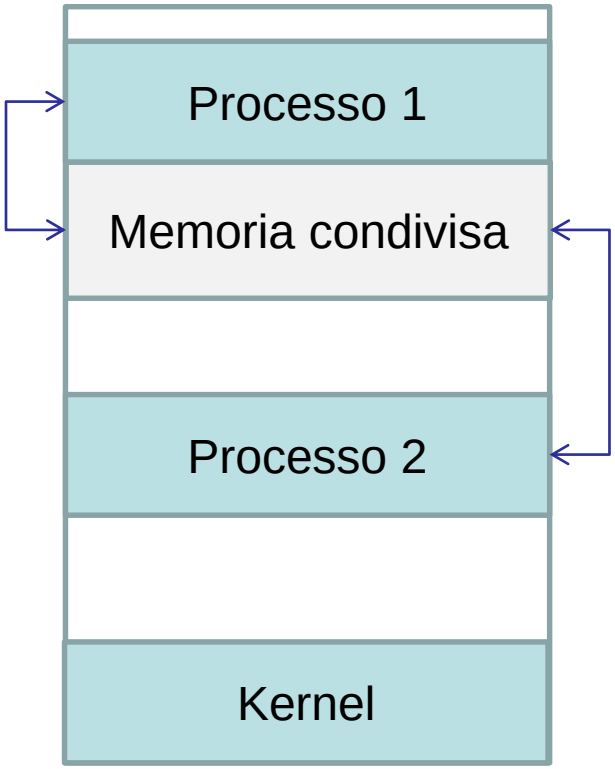
- Un processo è detto ***indipendente*** se non può influenzare o essere influenzato dagli altri processi in esecuzione nel sistema. Un processo che non condivide dati con altri processi è indipendente.
- Un processo è ***cooperante*** se può influenzare o essere influenzato da altri processi in esecuzione nel sistema. Chiaramente, qualsiasi processo che condivide dati con altri processi è un processo cooperante.
- Un sistema operativo che consente la cooperazione tra processi offre molti vantaggi, tra i quali:
  - ***Modularità***. Si può progettare e realizzare il sistema operativo in maniera modulare, suddividendo le funzioni di sistema in processi o thread separati, come descritto in una precedente lezione.
  - ***Condivisione delle informazioni*****.** Processi diversi potrebbero condividere le stesse informazioni (per esempio, un file condiviso).
  - ***Maggiore velocità di calcolo***. Per eseguire un'applicazione più velocemente, bisogna suddividerla in più attività, ciascuna delle quali sarà eseguita in parallelo con le altre. Tuttavia, un tale aumento di velocità può essere raggiunta solo se il computer dispone di più core.

- Ci sono due modelli fondamentali di comunicazione tra processi: *a **memoria condivisa*** **(shared memory)** e ***a scambio di messaggi (message passing)*****.**
- Nel primo modello, i processi allocano, mediante chiamate di sistema del kernel, zone di memoria condivise nelle quali possono scambiarsi informazioni.
- Tipicamente, uno dei processi esegue una chiamata di sistema per ottenere un'area di memoria condivisa; gli altri processi in seguito compiono chiamate per potervi accedere.
- Nel modello *a scambio di messaggi* , la comunicazione avviene tramite messaggi scambiati tra i processi cooperanti.
- L'insieme di chiamate di sistema del kernel, che permettono la comunicazione, è detto ***IPC, Inter Process Comunication***.
- Generalmente, nei sistemi operativi sono implementati entrambi i modelli.

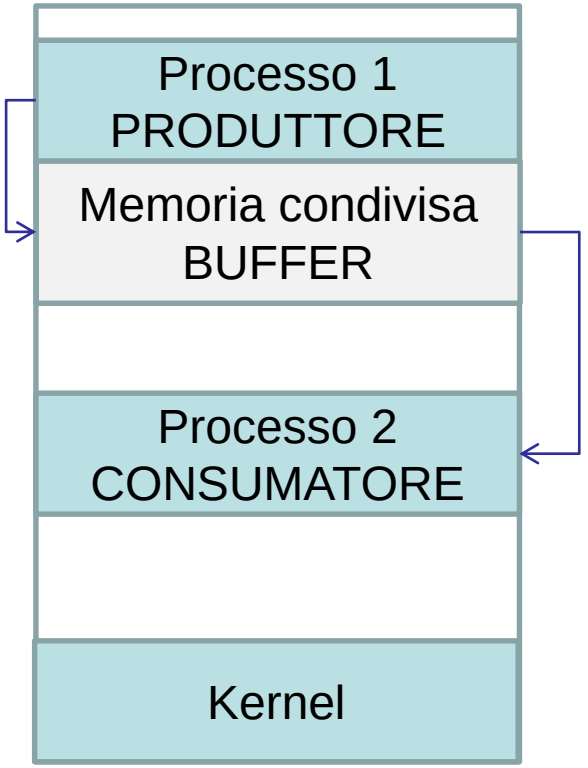
- La memoria condivisa consente velocità di trasferimento di dati più elevate rispetto allo scambio di messaggi. La maggiore efficienza in termini di velocità della memoria condivisa è dovuta al fatto che, una volta che i processi hanno ottenuto una zona di memoria comune, possono successivamente accedervi attraverso funzioni di libreria che girano nello spazio utente, senza eseguire chiamate del kernel. Quest'ultime devono invece essere usate ogni volta che un processo invia o riceve un messaggio.
- Tuttavia nelle architetture multi core, la comunicazione a scambio di messaggi fornisce prestazioni migliori rispetto alla memoria condivisa. Questo perché nelle architetture multiprocessore sono presenti molte cache, nelle quali i dati condivisi sono copiati creando problemi di coerenza. Tali problemi sono risolti con operazioni eseguite dal kernel, producendo quindi un considerevole overhead con conseguente diminuzione delle prestazioni.
- Poiché il numero di core di elaborazione sui sistemi aumenta di anno in anno, è possibile che lo scambio di messaggi diverrà la tecnica preferita per l'IPC.

Memoria condivisa

- Il modello di comunicazione tra processi a memoria condivisa si basa sull'allocazione di una parte di memoria condivisa.
- Per illustrare il concetto di processi cooperanti, consideriamo il problema del ***produttore-consumatore***, che è un paradigma comune per processi cooperanti.
- Un processo ***produttore*** produce informazioni che sono consumate da un processo ***consumatore***.
- Per consentire l'esecuzione parallela di processi produttori e consumatori, è necessario un buffer nel quale il produttore inserisce i dati che il consumatore preleva.
- Questo buffer risiederà in una area di memoria che è condivisa sia dal produttore che dal consumatore. Un produttore può produrre un dato mentre il consumatore ne consuma un altro. I processi produttore e consumatore devono essere sincronizzati, in modo che il consumatore non tenti di consumare un dato che non è stato ancora prodotto.

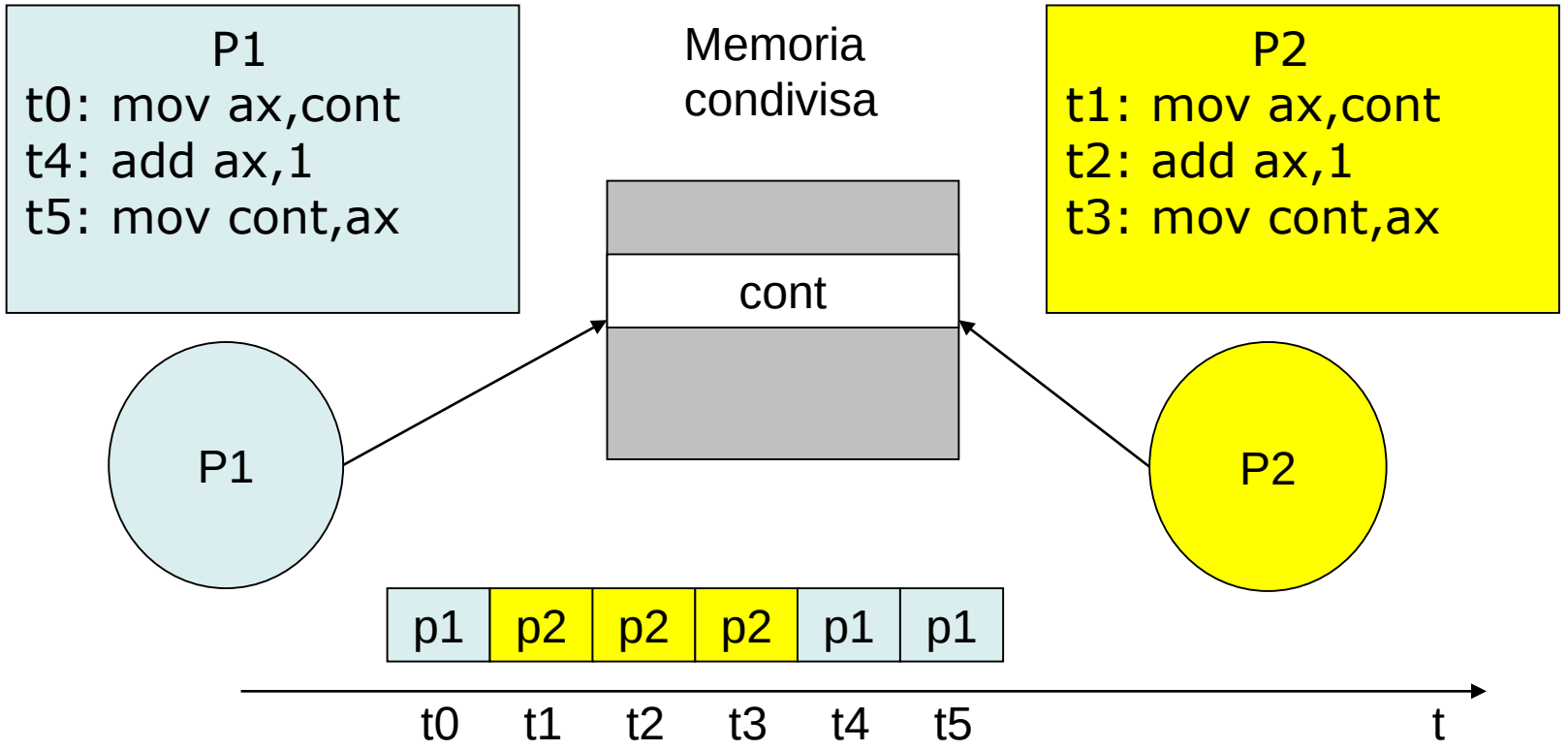


Modello memoria condivisa



Modello produttore-consumatore

- Il buffer può avere dimensione N (messaggi).
- Nel caso in cui il buffere ha dimensione N=1, il corretto funzionamento è dato dalla sequenza: inserimento - prelievo – inserimento - prelievo…
- Altre sequenze come ad esempio inserimento – inserimento - prelievo oppure inserimento – prelievo - prelievo porterebbero ad un funzionamento errato.
- Nel caso di processi cooperanti, le operazioni devono essere svolte seguendo un ordine preciso.
- La figura seguente mostra come si potrebbe produrre un errore nel caso in cui due processi che condividono una stessa variabile, ne modifichino il valore senza ricorrere ad alcuna tecnica di sincronizzazione. Nell'esempio, due processi **P1** e **P2** condividono una variabile intera **cont**, inizializzata a zero, che incrementano ogni volta che eseguono l'operazione **cont = cont + 1**. Si può notare che al termine delle operazioni svolte dai due processi il valore di cont è 1, invece del valore corretto che sarebbe 2.


P1
t0: mov ax,cont
t4: add ax,1
t5: mov cont,ax
Memoria
condivisa
P2
t1: mov ax,cont
t2: add ax,1
t3: mov cont,ax
cont
P1
P2
p1
p2
p2
p2
p1
p1
t0
t1
t2
t3
t4
t5
t

Memoria condivisa in POSIX

- In **POSIX** la memoria condivisa è gestita utilizzando file mappati in memoria, che associano a un file una regione di memoria condivisa.
- Un processo deve prima creare un segmento di memoria condivisa utilizzando la chiamata di sistema ***shm_open ()***:

```
shm_fd = shm_open (const char *name, int oflag,
          mode_t mode)
```

- Il primo parametro ***name*** specifica il nome del segmento di memoria condivisa. I processi che desiderano accedere a questa memoria condivisa devono fare riferimento a tale segmento con questo nome. Il parametro *oflag* specifica le modalità di accesso al segmento. che il segmento condiviso deve essere creato se non esiste ancora (O CREAT), e che è aperto per la lettura e la scrittura (O_RDRW).
- L'ultimo parametro definisce le autorizzazioni che il processo ha sul segmento condiviso.

- In caso di successo la *shm_open ()* restituisce un numero intero che identifica il descrittore del segmento di memoria condivisa.
- Una volta creato (o collegato) il segmento ha dimensione nulla. La funzione ***ftruncate ()*** può essere usata per dimensionare (in byte) il segmento.
- Infine, la funzione ***mmap ()*** crea un file mappato in memoria corrispondente al segmento condiviso. Inoltre, restituisce un puntatore al file che è utilizzato per accedere al segmento condiviso.
- I programmi di seguito mostrati utilizzano il modello produttore-consumatore in attuazione memoria condivisa.
- Il produttore crea un segmento di memoria condivisa nel quale scrive i suoi messaggi, e il consumatore legge dal segmento condiviso.
- Il produttore crea un segmento di memoria condivisa con nome /MEMCOND e vi scrive il messaggio "Saluti a tutti!".
- Il processo consumatore legge e visualizza il contenuto della memoria condivisa. Il consumatore chiama anche la funzione ***shm_unlink ()***, che rimuove il segmento di memoria condivisa.

Processo produttore

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>        /* costanti mode  */
#include <fcntl.h>           /* costanti O_*  */
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

int main(){
  const int SIZE=4096; // dimensione del segmento condiv.
  const char *nome = "/MEMCOND"; // nome del segmento
```

```
void *shm_ptr; // puntatore al segmento condiviso
/* dati scritti in memoria condivisa  */
const char *string_0 = "Saluti";
const char *string_1 = " a tutti!";
int shm_fd; // file descriptor del segmento
/* crea il segmento di memoria condivisa */
shm_fd = shm_open(nome, O_CREAT|O_RDWR, 0666);
ftruncate(shm_fd, SIZE); // dimensiona il segmento
/* memory map del segmento */
shm_ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED,
          shm_fd, 0);
sprintf(shm_ptr,"%s",string_0);// scrive nel segm.
shm_ptr += strlen(string_0);
sprintf(shm_ptr,"%s",string_1);
shm_ptr += strlen(string_1);
return 0; }
```

Processo consumatore

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>        /* costanti mode  */
#include <fcntl.h>           /* costanti O_*  */
#include <unistd.h>
#include <sys/types.h>
#include <string.h>

int main () {
  const int SIZE=4096; // dimensione del segmento condiv.
  const char *nome = "/MEMCOND"; // nome del segmento
  int shm_fd; // Descrittore del segmento condiviso
```

```
  void * shm_ptr; // Puntatore al segmento condiviso
  /* Accesso al segmento in lettura  */
  shm_fd = shm_open(nome, O_RDONLY, 0666);
  /* memory map del segmento */
  shm_ptr = mmap (0, SIZE, PROT_READ, MAP_SHARED, shm_fd,
                  0);
  /* Lettura dal segmento di memoria condivisa */
  printf ("% s", (char *) shm_ptr);
  /* Rimozione del segmento di memoria condivisa */
  shm_unlink (nome);
  return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/stat.h>        /* costanti mode  */
#include <fcntl.h>           /* costanti O_*  */
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
int N=5;
Char nome[][20]={"Pietro","Antonio","Laura","Luisa",
                 "Lino"};

struct buffer_t {
       int id;
       char text[64];
} *buffer;
```

```
// processo produttore

void produttore(){
  int i;
  for (i=0;i<N;i++) {
    buffer->id=i;
    strcpy(buffer->text,nome[i]);
    printf ("msg scritto: %d %s\n",buffer->id,
             buffer->text);
    usleep(200);
  }
}
```

```
// processo consumatore

void consumatore(){
  int i;
  for (i=0;i<N;i++) {
    printf ("msg letto: %d %s\n",buffer->id,
            buffer->text);
    usleep(200);
  }
}
```

```
int main(){
  pid_t pid;
  int shm_id, SIZE;
  shm_id=shm_open("/memcond",O_CREAT|O_RDWR,0666);
  SIZE=sizeof(struct buffer_t);
  ftruncate(shm_id,SIZE);
  buffer=mmap(0,SIZE,PROT_READ|PROT_WRITE,
         MAP_SHARED,shm_id,0);
  pid=fork();
  if (pid==0) {
    produttore();
  } else {
      consumatore();
      shm_unlink("/memcond");
      wait(NULL);
  }
}
```