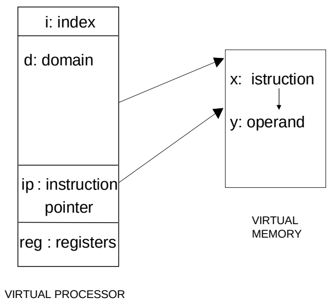
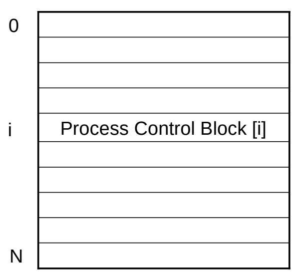
Module 5: Process Synchronization

- Process management
- Event flags
- Semaphore Variables and primitives
- Classical synchronization problems
- Examples of process and threads synchronization in UNIX

Process Managment



Process List



Process Control Block

Process status word	
status	PL (i)
priority	
link	
other system information	

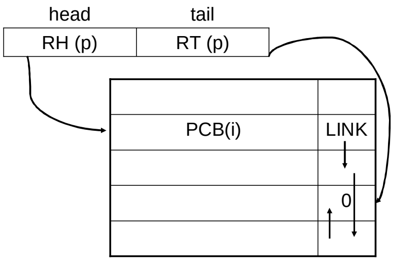
Context Switching

CP is the identifier of the running process

```
SAVESW : copy (D,IP,REGS) into PL(CP)
LOADSW : load (D,IP,REGS) from PL(CP)

SWITCH:
     SAVESW
     CP = index of the new ready process
     LOADSW
     dispatch
```


Ready List

RL (p) : Ready list of p priority



Round Robin

PRI is the priority register
P = 0 is the maximum priority

```
if (TIMER == 0) {
  LINK(RT(PRI))= CP;
  RT(PRI)= CP;
  RH(PRI)= LINK(RH(PRI));
  LINK(CP)= 0;
  SWITCH;
}
```

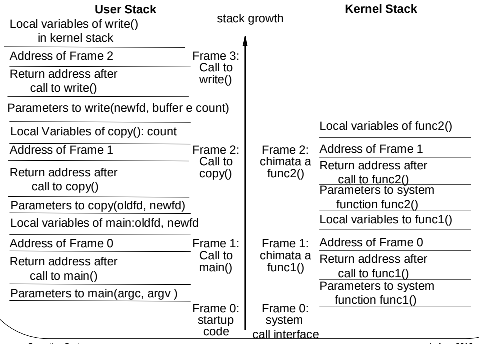

User and kernel stacks – file copy

```
/* copy.c */

#include <stdio.h>
#define BUFDIM 1024
char buffer[ BUFDIM ];
main( int argc, char ** argv ){
  int oldfd, newfd; /* file descriptors */
     .
```


User and kernel stacks – file copy

```
    .
    .
    .
copy( oldfd, newfd );
  exit( 0 );
}

copy( int oldfd, int newfd ){
  int count;
  while ((count = read( oldfd, buffer,
            BUFDIM)) > 0 )
    write( newfd, buffer, count );
}
```

User and Kernel Stack



Event flag

Process synchronization by means of signals

```
Event e;

Sender:
while (TRUE) {
  ...
  cause e;
}

Receiver:
while (TRUE) {
  await e;
  ...
}
```


Event flag

Process synchronization by means of signals

Case 1	Case 2
R not waiting	R not waiting
(R) AWAIT e	(S) CAUSE E
R waiting	R not waiting
(S) CAUSE e	(R) AWAIT E
R not waiting	R waiting

Event signals in Unix

LOCK:

while (condition)

sleep (event: condition false);

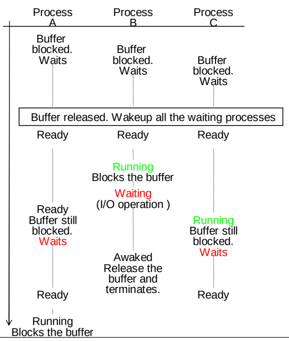
condition = TRUE;

UNLOCK:

Condition = FALSE;

wakeup (event: condition is false);

Event signals in Unix



Semaphore Variables

- A semaphore can be defined by this structure:

```
typedef struct semaphore_tag {
        char lock;
        int cnt;
        process_t *head;
} semaphore_t;
```

- Let's define two kernel operation:
  - **`disable(S)`** - suspends the issuing process, CP, and appends it to the semaphore list **`S;`** **`unlock(S.lock)`**.
  - **`enable(S)`** – moves the first process blocked in the semaphore **`S`** list to the ready list.

disable(S) Implementation

Count	Head	Tail
CNT(J)	SH(J)	ST(J)

Index in Process List

```
RH[PRI]= LINK[CP];
STATO[CP] = J;
if (SH[J]]== 0)
  SH[J] = CP;
else
  LINK[ST[J]]= CP;
LINK[CP]= 0;
ST[J] = CP;
while (RH[PRI]==0) PRI++;
SWITCH; // con CP=RH[PRI]
```


enable(S) Implementation

```
I = SH[J]
SH[J] = LINK[I]
STATO[I] = 0;
p = priority[I];
if (RH[p] == 0)
  RH[p] = I;
else
  LINK[RT[p]] = I;
LINK[I] = 0;
RT[p] = I;
if p < PRI {
  PRI = P;
  SWITCH  // with CP=RH[PRI]
}
```


Semaphore Primitives

WAIT(*S*)
{
  lock(S.lock);
  S.cnt --;
  if (*S.cnt* < 0)
    disable(S);
  unlock(S.lock);
}

SIGNAL(*S*)
{
  lock(S.lock);
  S.cnt++;
  if (*S.cnt* ≤ 0)
    enable(S);
  unlock(S.lock);
}

INIT(S,k) { S.cnt = k}   k >= 0

Look !

Synchronization by means of Semafores

B must be executed in P_k only after *A* has been executed by P_j

INIT(S) = 0; // pure synchronization

P_i

⋮

A

SIGNAL(S) → WAIT(S)

B

⋮

P_k

Client-Server Synchronization

INIT(call) = 0; INIT(ret) = 0;

P_i

⋮

prepares data

SIGNAL(call)

WAIT(ret)

Process the results

⋮

P_k

while(TRUE) {

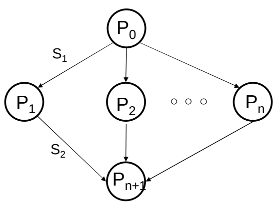
WAIT(call)

data processing

SIGNAL(ret)

}

Cobegin-Coend implementation with semaphores



Cobegin-Coend implementation with semaphores

INIT(S1) = 0; INIT(S2) = 0;

P_0	P_i	P_{n+1}
Body of P_0 ⋮ for(i=1;i<=n;i++) SIGNAL(S1) ⋮	WAIT(S1) Body of P_i ⋮ SIGNAL(S_2)	for(i=1;i<=n;i++) WAIT(S2) Body of P_{n+1}

Mutual exclusion with semaphores

INIT(ME) = 1;   // allows the access of a process
// to its critical region

P_i

WAIT(ME)

Critical Region$_i$

SIGNAL(ME)

Time-dependent Errors

INIT(S) = 1;

```
(* P1 *)
begin
 signal( S );
  CR1
 wait( S );
end
```

```
(* P2 *)
begin
 wait( S );
  CR2
 signal( S );
end
```

```
(* P3 *)
begin
 wait( S );
  CR3
 signal( S );
end
```

Time-dependent Errors

INIT(S) = 1;

```
(* P1 *)          (* P2 *)          (* P3 *)
begin             begin             begin
 wait( S );         wait( S );        wait( S );
  CR1               CR2               CR3
 wait( S );        signal( S );      signal( S );
end               end               end
```

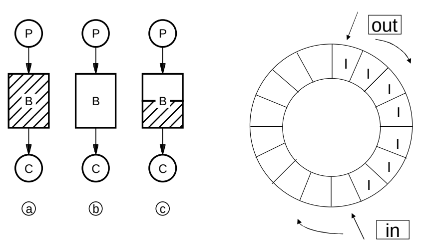

Deadlock and Starvation

- **Deadlock** – two or more processes wait indefinitely an event that can b e cause only by one of the blocked processes
- INIT(S) = 1; INIT(Q) = 1;

P_0	P_1
WAIT(S);	*WAIT*(Q);
WAIT(Q);	*WAIT*(S);
⋮	⋮
SIGNAL(Q);	*SIGNAL*(S);
SIGNAL(S)	*SIGNAL*(Q);

- **Starvation** – perpetual block. A process is never moved to the ready list from the list of the semaphore on which it is blocked.

Producer-Consumer



N = MAX elements

Access functions to the common buffer

```
Message buffer [MAX];
int in, out;
```

```
init(){
  in = 0;
  out = 0;
}
```

```
enter(Message m ){
  buffer[in] = m;
  in=(in + 1)% MAX;
}
```

```
remove (Message m){
  m = buffer[out];
  out=(out + 1) %MAX;
}
```


Producer-Consumer

INIT(full) = 0; INIT(empty) = MAX;

```
Producer(){
Message m;
while (TRUE) {
    produce m;
    wait( empty );

      enter( m );

    signal( full );
 }
}
```

```
Consumer(){
Message m;
while (TRUE) {
    wait( full );

      m = remove();

    signal( empty );
    consume m;
 }
}
```


Producer-Consumer

INIT(full) = 0; INIT(empty) = MAX;

INIT(ME_p) = 1; INIT(ME_c) = 1;

```
Producer(){
Message m;
while (TRUE) {
    produce m;
    wait( empty );
    wait( ME_p );
      enter( m );
    signal( ME_p );
    signal( full );
 }
}
```

```
Consumer(){
Message m;
while (TRUE) {
    wait( full );
    wait( ME_c );
      m = remove();
    signal( ME_c );
    signal( empty );
    consume m;
 }
}
```

Synchronization Examples of UNIX Processes

semaphore.c
- Mutual exclusion by means of semaphores implemented with pipe()

voila-alarm.c
- Alarm clock signals

scheduler.c
- Scheduler of processes running at fixed intervals of time

POSIX semaphores

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t *sem);         /* P(sem), wait(sem) */
int sem_post(sem_t *sem);         /* V(sem), signal(sem) */

int sem_getvalue(sem_t *sem, int *sval);
int sem_trywait(sem_t *sem);

int sem_destroy(sem_t *sem);      /* undo sem_init() */

/* named semaphores - these are less useful here */
sem_t *sem_open( ... );
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
```


Synchronization Examples of UNIX Threads

mutex.c
- Mutual exclusion by means of mutex

trylock.c
- Mutual exclusion with mutex testing

cond.c
- Event management with condition variables

prod-cons.c
- Producer-consumer