Process Creation and Control

Computer Architecture & OS Lab

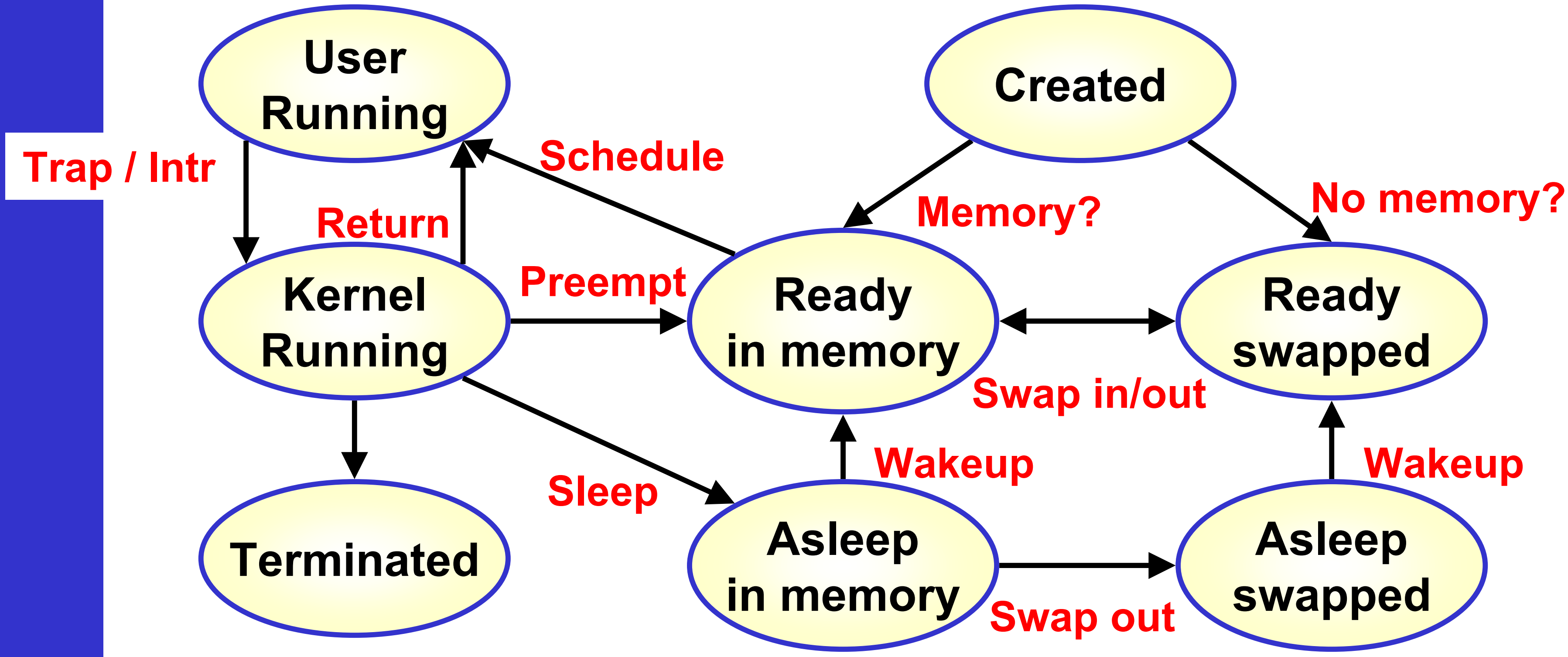
Dept. of Computer Science & Engineering

Indian Institute of Technology, Kharagpur

Process

- A process is a program in execution
- Contents:
  - Process control block
    - Process identification
    - Process state information
    - Process control information
  - User stack
  - Private user address space (program, data)
  - Shared address space

Process State Transitions


User Running

Created

Trap / Intr

Schedule

Return

Memory?

No memory?

Kernel Running

Preempt

Ready in memory

Ready swapped

Swap in/out

Wakeup

Wakeup

Sleep

Terminated

Asleep in memory

Asleep swapped

Swap out

How to create a new process?

- The fork() system call
  - It creates a new process as a *child process* of the calling process (*parent*)
  - Both have similar code segments
  - The child gets a copy of the parents data segment at the time of forking

- How can the child realize that it is the child and not the parent?
- How can we make the child and parent do different things?

The return value of fork()

- fork() returns a value to both parent and child
  - The parent receives the process id of the child
  - The child receives 0 (zero)

Key idea:

```
if (fork() == 0)
      { /* I am the child process */ }
else
      { /* I am the parent process */ }
```

The first program: fork1.c

```
#include <stdio.h>
#include <sys/ipc.h>
main( )
{
    if (fork( ) == 0) {        /* Child */
          while (1) {          for (i=0; i<100000; i++) ;
                               printf("\t\t\t Child executing\n ");
          }
    }
    else {                     /* Parent */
          while (1) {          for (i=0; i<100000; i++) ;
                               printf("Parent executing\n");  }
    }
}
```

Waiting for child termination

- The parent process can wait for the child process to terminate using the call:

```
waitpid( pid, NULL, 0 )
```

-- where pid is the identifier of the child process (returned by fork( ))

-- what are the other two parameters?

The second program: fork2.c

```
#include <stdio.h>
#include <sys/ipc.h>
main()
{
    int i, x = 10, pid1, pid2 ;
    printf("Before forking, the value of x is %d\n", x);

    if ( ( pid1 = fork( ) ) == 0) {    /* First child process */
          for (i=0 ; i < 5; i++) {
             printf("\t\t\t At first child: x= %d\n", x);
             x= x+10;        sleep(1) ; /* Sleep for 1 second */
          }
    }
```

The second program: fork2.c

```
   else {         /* Parent process */

       if ( ( pid2 = fork( ) ) == 0) {  /* Second child */
          for (i=0 ; i < 5; i++) {
             printf("\t\t\t\t\t\t At second child: x= %d\n", x);
             x= x+20;  sleep(1) ; /* Sleep for 1 second */
          }
       }
       else {  /* Parent process */
             waitpid(pid1,NULL,0);
             waitpid(pid2,NULL,0);
             printf("Both children terminated\n");
       }
}}
```

Points to ponder: fork3.c

```
#include <stdio.h>
#include <sys/ipc.h>
main( )
{
    int x=0, pid;
    printf("Hello!");

    if ( ( pid = fork() ) == 0) {   /* Child */
      printf("\nChild:\t Address of x: %x\t
                              Value of x: %d \n", &x, x);
      x = 20;
      printf("Child:\t Address of x: %x\t
                              Value of x: %d \n", &x, x);
    }
```

Points to ponder

```
    else {          /* Parent */
      waitpid(pid, NULL, 0);
      printf("\nParent:\t Address of x: %x\t
                               Value of x: %d \n", &x, x);
      x = 10;
      printf("Parent:\t Address of x: %x\t
                               Value of x: %d \n", &x, x);


    }
}
```

- Why is Hello! printed twice?
- Though the address of x is the same in the parent and in the child, they contain different values. Hows this possible?

Shared Memory

Computer Architecture & OS Lab

Dept. of Computer Science & Engineering

Indian Institute of Technology, Kharagpur

Shared Memory System Calls

- Creation:

```
int shmid = shmget( IPC_PRIVATE,
                     <no of bytes>, 0777|IPC_CREAT )
```

  - This call creates the shared memory segment and returns its identifier

- Attach:

```
char * shmat( shmid, 0, 0 )
```

  - This call attaches the shared memory segment with the logical address space of the calling process and returns the logical address

Using shared memory: shm.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>

main()
{
    int shmid, *a, *b, i;

    /* Acquire a shared array of 2 integers */
    shmid = shmget( IPC_PRIVATE,
                      2*sizeof(int), 0777|IPC_CREAT);
```

Using shared memory: shm.c

```
  if ((pid = fork()) == 0) {            /* Child */
        b = (int *) shmat( shmid, 0, 0 );   /* Attach to child */
        for( i=0; i< 10; i++) {
                  sleep(1);
                  printf("\t\t\t Child reads: %d,%d\n",b[0],b[1]);
  }     }
  else {                           /* Parent */
        a = (int *) shmat( shmid, 0, 0 );  /* Attach to parent */
        a[0] = 0; a[1] = 1;
        for( i=0; i< 10; i++) {
                  sleep(1); a[0] = a[0] + a[1]; a[1] = a[0] + a[1];
                  printf("Parent writes: %d,%d\n",a[0],a[1]);
        }
        waitpid( pid );
}}
```

Assignment: Concurrent Mergesort

Write a program for mergesort that works as follows.

- The given set of integers is stored in shared memory.
- If the number of integers is less than 20, then the process sorts the integers using bubble-sort.
- Otherwise, it recursively creates one child process for sorting the left half and another child process for sorting the right half.
- After both children terminate, the parent merges the left and right halves.
- Note that the child processes should in turn follow the same procedure and create children of their own if needed.

Your program should read a list of integers (and nothing else). The first integer in the list will indicate the number of integers to be read.