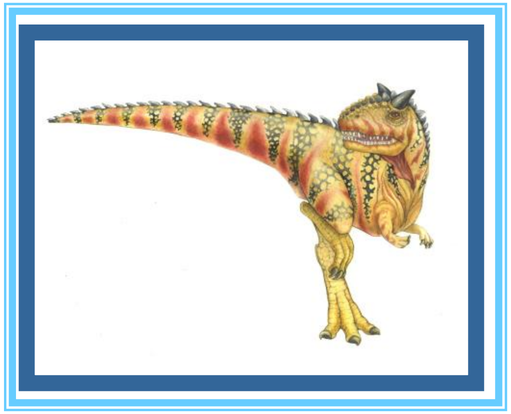
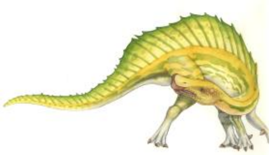
Chapter 3: Processes

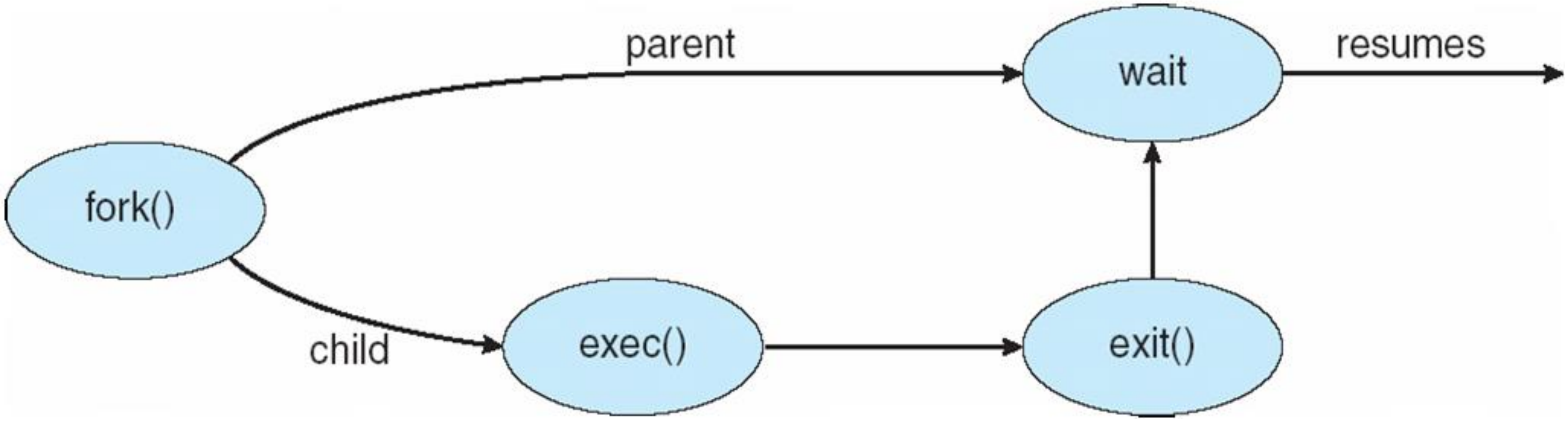


Process Creation

```
/* What’s the output???? */
int main()
{
        pid_t  pid;
        int i;
        /* fork another process */
        pid = fork();
        fprintf(stderr,”The value: %d”, value);
        if (pid < 0) { /* error occurred */
         fprintf(stderr, "Fork Failed");
         exit (1);
        }
        else if (pid == 0) { /* child process */
         for (i = 1; i <= 2; ++i) {printf (“%d”, -i);}
        }
        else { /* parent process */
         for (i = 1; i <= 2; ++i) {printf (“%d”, i);}
         wait (NULL); /* parent will wait for the child to complete */
         printf ("Child Complete");
         exit (0);
        }
 printf ("Child Complete");
```



Process Creation



Process Termination

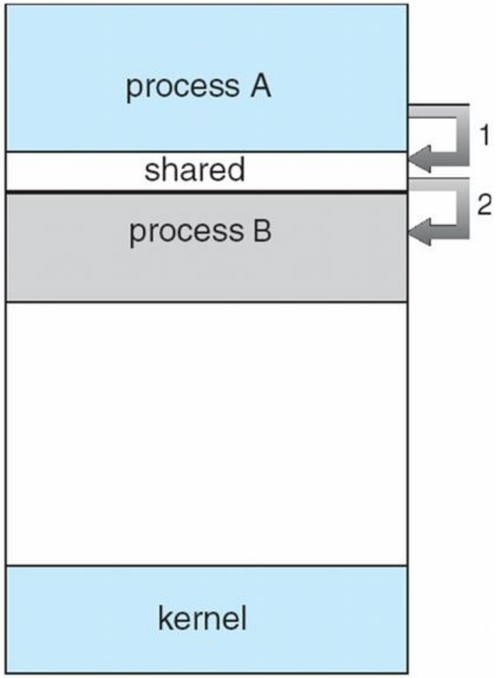
- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)

- Cascading termination
  - kill -9 pid

Interprocess Communication (IPC)

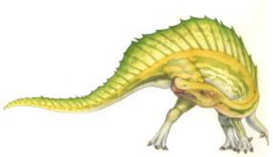
- Why do we want this?

- Two models of IPC
  - Shared memory - establish shared memory and treat all accesses as routine memory accesses



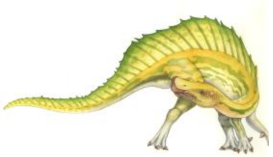
Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience



Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer
  - *bounded-buffer* assumes that there is a fixed buffer size



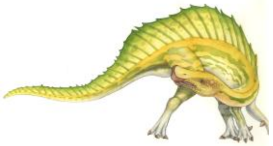
Bounded-Buffer – Shared-Memory Solution

- Shared data

```
#define BUFFER_SIZE 10
typedef struct item {
  . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements



Bounded-Buffer – Producer

```
while (true)
{
   /* Produce an item */
      while (((in + 1) % BUFFER_SIZE)  == out)
     {   /* do nothing -- no free buffers */}
    buffer[in] = item;
    in = (in + 1) % BUFFER_SIZE;
   }
```

Bounded Buffer – Consumer

```
while (true) {
       while (in == out)
        { /* do nothing -- nothing to consume */}

     // remove an item from the buffer
     item = buffer[out];
     out = (out + 1) % BUFFER SIZE;
     return item;
  }
```