Process

Processes

- Processes are the fundamental structure of operating systems
  - A unit of activity characterized by a sequential thread of execution, current state, and an associated set of system resources
- Processes solved the problems introduced by
  - Multiprogramming batch operations
  - Time sharing
  - Real-time transaction systems

Processes (continued…)

- Problems in Coordination of processes
  - Improper synchronization
  - Failed mutual exclusion
  - Non-determinate program operation
  - Deadlocks
- Processes consist of three components
  - An executable program
  - Associated data (variables, workspace, buffers, stacks, etc.)
  - The execution context of the program

Process Concepts

- Process Definition
- Process Creation
- Process States and State Transitions
- Important Information Associated with Processes

What is a Process (or Task)?

- Sequence of instructions that executes
  - The entity that can be assigned to and executed on a processor
  - A unit of activity characterized by a single sequential thread of execution
  - Can be traced
- Associated data needed by the program
- Context
  - All information the operating system needs to manage the process
  - A current state and an associated set of system resources

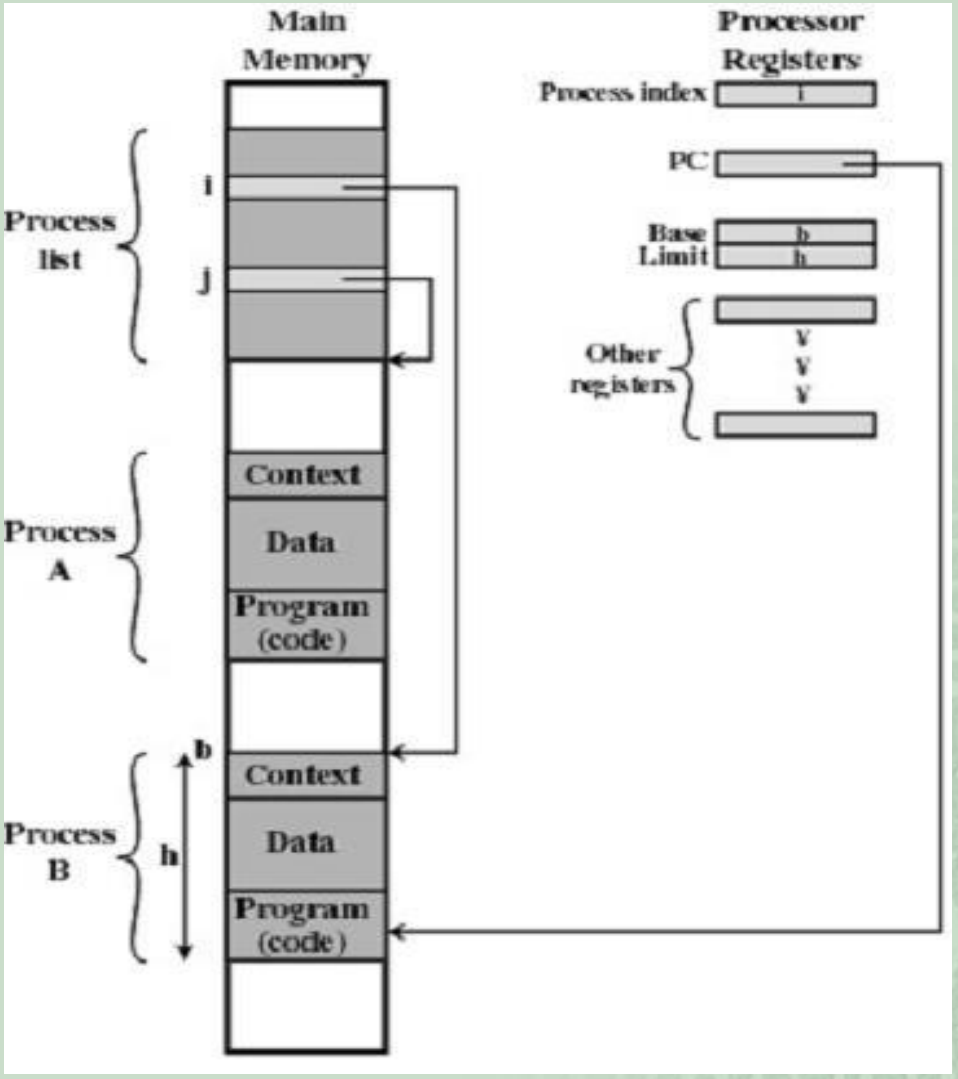
‘program’ versus ‘process’

- A ‘program’ is a sequence of instructions
- It’s an algorithm, expressed in a computer language, but it does nothing on its own
- A ‘process’ is a succession of actions that are performed when a program executes
- It’s a dynamically changing entity, which has an existence over time, but it requires a processor and it uses various resources

What is Required of an OS?

- Assist the execution of a process
  - interleave the execution of several processes
  - maximize processor utilization
  - provide reasonable response time
- Allocate resources to processes
  - fairness
  - avoid starvation / deadlock
- Support interprocess activities
  - communication
  - user creation of processes

Process Implementation



Process Trace

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.2 Traces of Processes of Figure 3.1

1	5000	27	12004
2	5001	28	12005
3	5002	----------	Time out
4	5003	29	100
5	5004	30	101
6	5005	31	102
----------	Time out	32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	----------	Time out
16	8003	41	100
----------	I/O request	42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		----------	Time out

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.3 Combined Trace of Processes of Figure 3.1

Process Concepts

- Process Definition
- Process Creation
- Process States and State Transitions
- Important Information Associated with Processes

What Initiates Process Creation?

- Submission of a batch job
- User logs on
- Created to provide a service such as printing
- Process creates another process
  - Modularity
  - Parallelism
  - Parent – child relationship
- Deciding how to allocate the resources is a policy that is determined by the OS

Process Creation Decisions

- Resource Allocation
  - Treat as a new process
  - Divide parent's resources among children
- Execution
  - child runs concurrently with parent
  - parent waits until some or all children terminate
- Address Space
  - copy of parent
  - new program loaded into address space

Example: Unix Process Creation

- A new process is created by the *fork* call
- Child and parent are identical
  - child returns a 0
  - parent returns nonzero
- Both parent and child execute next line
- Parent and child processes can execute different code based upon the *fork* return values

Unix Example

```
switch (child1 = fork())
{   case -1:  printf("Can't fork");
              exit(-1);
    case 0:   child1_function(…);          // child 1
              exit(0);
    default:  switch (child2 = fork())           // parent
              {   case -1:  printf("Can't fork");
                            exit(-1);
                  case 0:   child2_function(…); // child 2
                            exit(0);
                  default:  // Wait for child two
                            waitpid(child2, status2, options);
              }
              /* Wait for child one */
              waitpid(child1, status1, options);
              fflush(stdout);
}
```

Windows NT process creation

- Offers the fork-exec model
- Process created by CreateProcess call
- Child process executes
  - concurrently with parent
  - parent must wait
- CreateProcess loads a program into the address space of the child process

Windows NT Example

```
// Declarations.

   PROCESS_INFORMATION pi;
   STARTUPINFO si;

// Set up members of STARTUPINFO structure.
   ZeroMemory( &si, sizeof(STARTUPINFO) );
   si.cb = sizeof(STARTUPINFO);

// Create the child process.
   CreateProcess(NULL,
      program,        // command line
      NULL,           // process security attributes
      NULL,           // primary thread security attributes
      TRUE,           // handles are inherited
      0,              // creation flags
      NULL,           // use parent's environment
      NULL,           // use parent's current directory
      &si,             // STARTUPINFO pointer
      &pi);            // receives PROCESS_INFORMATION
```

Reasons for Process Termination

- User logs off
- Normal completion
- Batch issues *Halt*
- Memory unavailable
- Bounds violation
- Protection error
  - example write to read-only file
- Arithmetic error
- Time overrun
  - event timeout
- I/O failure
- Invalid instruction
  - tried to execute data
- Privileged instruction
- Data misuse
- Operating system intervention
  - such as when deadlock occurs
- Parent terminates so child processes terminate
- Parent request