List of Slides

1  Processes and Process Control
4  Definitions of A Process
5  Systems state vs. Process State
6  A 2 State Process Model
7  Process Creation, what does it mean?
8  Process Termination’s meaning
10 A Five State Model of Process Control
11 Queueing Models of Systems having I/O
12 Swapping (Suspending) Processes
13 CPU with I/O and Suspended State
14 Operations on Processes
15 O/S support for Processes
16 O/S Global Process Structures
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Memory Tables</td>
</tr>
<tr>
<td>18</td>
<td>I/O Tables</td>
</tr>
<tr>
<td>18</td>
<td>File Tables</td>
</tr>
<tr>
<td>18</td>
<td>Process Tables</td>
</tr>
<tr>
<td>19</td>
<td>Process Image</td>
</tr>
<tr>
<td>20</td>
<td>Process Control Blocks</td>
</tr>
<tr>
<td>21</td>
<td>Process Memory Management</td>
</tr>
<tr>
<td>22</td>
<td>O/S Use of PCBs</td>
</tr>
<tr>
<td>23</td>
<td>O/S Kernel Services</td>
</tr>
<tr>
<td>24</td>
<td>Process Switching</td>
</tr>
<tr>
<td>25</td>
<td>Running the O/S</td>
</tr>
<tr>
<td>26</td>
<td>Processes and Threads</td>
</tr>
<tr>
<td>27</td>
<td>Thread Management in Windows NT</td>
</tr>
<tr>
<td>28</td>
<td>Examples — MVS</td>
</tr>
<tr>
<td>29</td>
<td>MVS List Structures</td>
</tr>
<tr>
<td>30</td>
<td>Examples — Unix</td>
</tr>
<tr>
<td>32</td>
<td>4.4 BSD Unix Scheduling</td>
</tr>
</tbody>
</table>
33  4.4 BSD Process Structure
34  4.4 BSD Process Entry
35  4.4 BSD Process Group
36  4.4 BSD Process Group
37  4.4 BSD Process States
38  Examples — Windows NT
39  Hardware Support for Debugging
40  Unix Support for Debugging
42  Ptrace — Unix Support for Debugging
43  Ptrace Functionality
44  Unix Procs file system
2 – Definitions of A Process

Deitel [1] offers the following definitions:

1. A program in execution
2. An asynchronous activity
3. the “animated spirit” of a procedure
4. the “locus of control” of a program in execution
5. that which is manifest by the existence of a process control block in the O/S
6. that entity which is assigned to processors
7. the “dispatchable” unit
3 – Systems state vs. Process State

1. Some processes are resident in memory.
2. The kernel is always resident.
3. One process runs at a time.

Figure 1: System Memory Layout
CPU is the only resource for many jobs.

1. Enter — Process Creation
2. Dispatch — Scheduling/Queueing Discipline
3. Pause — Give other jobs a chance
4. Exit — Process Termination

(a) State transition diagram

![State transition diagram](image)

(Figure 2: State Diagram of CPU Only System)
5 – Process Creation, what does it mean?

Some reasons for process creation include:

1. A new batch job
2. Interactive O/S login
3. O/S created
4. Spawned by an existing process

The frequency of process creation reflects the expense of creation.
6 – Process Termination’s meaning

Some reasons for process termination include:

1. Normal Completion

2. Excessive Resource Use
   (a) CPU Time Out
   (b) Insufficient Memory
   (c) File system full error

3. Security Violations/Programmer Errors
   (a) Illegal Address
   (b) Illegal Instruction
   (c) Privileged Instruction
   (d) Data Misuse (type error/initialization error)
4. Systems Control
   (a) Parent Job Terminated
   (b) Terminated by Parent
   (c) Operator or O/S intervention

The frequency of process creation reflects the expense of creation.
7 – A Five State Model of Process Control

Blocking — When a process waits on a non-cpu service (typically I/O).

Real systems have I/O, so a more realistic model is:

![State Diagram of CPU and I/O system](image)

Figure 3: State Diagram of CPU and I/O system
8 – Queueing Models of Systems having I/O

An architecture with I/O and CPU is:

(a) Single blocked queue

(b) Multiple blocked queues

Figure 4: Queueing Model of CPU and I/O system
9 – Swapping (Suspending) Processes

Suspending a Process — When the O/S saves the state of a nonrunning program from main memory to auxiliary memory.

Activation — When the O/S reloads a suspended process into main memory from auxiliary memory.

Some reasons for swapping out processes from memory:

1. System malfunction — Save state and resume after fix

2. User suspicious about partial results — Debugging/checkpointing

3. Correct short term load (or memory requirement) fluctuations

4. Fairness (one big process prevents others from running)
We can either treat a suspension as independent from blocking, or as mutually exclusive.

(a) With one suspend state

(b) With two suspend states

Figure 5: O/S supporting Blocking and Suspension
Deitel [1] lists some operations on processes:

1. Create a process
2. Destroy (terminate) a process
3. Block a process
4. Suspend a process
5. Resume (activate) a process
6. Change a process’s priority
7. Wake up a process
8. Wake up a process (put it into the ready state)
9. Enable a process to communicate with another process (interprocess communication)
12 – O/S support for Processes

The O/S in its role as resource manager and as run time interface controls resource access by mapping processes to resources.

Figure 6: Processes and Resources
13 – O/S Global Process Structures

The big picture for process management looks like:

![Diagram of process and resource structures]

Figure 7: Processes and Resources
14 – Memory Tables

Memory tables record the following information:

1. The allocation of main and auxiliary memory to processes

2. Memory protection (O/S vs. users, users from each other, read only vs. writable, instructions vs. data)

3. Control information for the virtual memory manager
15 – I/O Tables

Manage hardware control (and perhaps higher level control) of channels and peripherals in the system.

16 – File Tables

Provide security, access control, and naming support for persistent objects.

17 – Process Tables

Manages each individual process’s data structures, stores security permissions, and process state info.
Process Image — The state information (attributes) of the process (data/stack/instructions, I/O state). Process images typically contain:

1. User Data
2. User Program (Instructions)
3. System Stack
4. Process Control Block
Process Control Blocks (PCBs)— The data structures the O/S allocates for managing each process, containing:

1. Process identifiers — process id, parent id, user id.

2. Process State Info — User Visible Registers, Control Registers (and PC), and stack pointer.

Each process image is allocated its own virtual memory.

**Figure 8: Process Memory management**
21 – O/S Use of PCBs

The system accesses processes via their PCBs for state transitions and scheduling as per state diagram and queueing models.

Process Lists — Correspond to queues and service structures in queueing model.

Figure 9: O/S use of PCBs in Process Lists
22 – O/S Kernel Services

An O/S kernel provides privileged access to system resources, running in systems mode, control mode, or kernel mode.

Typical Kernel services include:


2. Memory Management — Allocation of address space to processes, Swapping, virtual memory management.

3. I/O Management — Buffer management, device and channel allocation to processes

4. Systems Support — Interrupt handling, Accounting, Monitoring.
23 – Process Switching

Process switching is the O/S transferring control from one process to another.

Issues include:

1. When to switch? Preemption vs. Non Preemption

2. Context Switching — done as follows:
   (a) Preserve the running process’s state in the PCB, if swapping save image.
   (b) Load process image’s PCB into system registers and memory restore the program counter.
The following are typical of O/S run time support structures:

1. Nonprocess O/S — A more primitive structure (MS-DOS, CP/M)

2. Single separate Kernel Process — A more monolithic approach, has the efficiency advantage of fewer context switches, but less flexible (macrokernel?). (Unix/Linux, Mac OS, VM, MVS).

3. System services via Kernel and User processes — A more flexible approach (microkernel?). (Windows NT, OS/2, MACH, GNU HURD, Amiga DOS).
Some people consider threads as a special form of process.

1. Processes control a unit of resource ownership.

2. A process is typically the unit of dispatching.

3. Threads share process context,

4. Threads are asynchronous,

5. Threads have less context than processes,

6. Threads can be created/terminated at a lower cost.

7. Threads cooperate to do a process in parallel with (relatively) fine granularity of parallelism

Threads are suited to shared memory SMP machines.
26 – Thread Management in Windows NT

Threads typically cooperate to do the same work as a traditional process. Often the system services are done in user space in a microkernel system (to make them run time configurable/flexible with an efficiency penalty).

Figure 10: Cooperation in a Typical Thread System (NT)
MVS has 3 task (process) states: Ready, Active, Waiting. Entire task may be swapped to auxiliary storage.

Consider a task composed of:

1. a main program,
2. a customer inquiry module,
3. an order entry module and
4. a production tracking module

Figure 11: MVS Address Space Example
MVS tracks system resources used and tasks using list structures.

**Figure 12: MVS List Structures Example**

- **SRB** = Service Request Block
- **ASCB** = Address Space Control Block
- **ASXB** = Address Space Extension Block
- **TCB** = Task Control Block
- **SQA** = System Queue Area
- **LSQA** = Local System Queue Area
- **RCT** = Region Control Area
- **STC** = Started Task Control
- **INIT** = Initiator
Unix has a macrokernel, and processes (some versions now have thread support).
Figure 13: Unix Process States
4.4 BSD implements a priority scheduler which

1. favors interactive processes over batch jobs.

2. is time sliced (completing a time slice lowers priority).

3. detects insufficient memory conditions and swaps out the least recently run process.
4.4 BSD (the basis for FreeBSD) has support for variable weight threads, where the kernel keeps track of each thread.

Show Figure McKusick Figure 4.1 Here

Figure 14: 4.4 BSD Unix Process States
Each process entry contains:

1. Process identification — pid and parent pid
2. Scheduling — priority, user-mode scheduling, CPU utilization and time spent sleeping
3. Process State — *Runnable, Sleeping, Stopped* and the *wait channel* describing the event it is waiting on, and a string describing the event.
4. Signal state — pending signals, signal masks and actions.
5. Tracing (for debugging)
6. Machine State
7. Timers — Real time and CPU.
33 – 4.4 BSD Process Group

The process group substructures contain:

1. Process Group Identification
2. User Credentials
3. Memory Management
4. File Descriptor
5. Resource accounting (and limits)
6. Statistics (e.g. profiling info)
7. Signal actions
8. User Structure
34 – 4.4 BSD Process Group

The process group substructures contain:

1. Process Group Identification
2. User Credentials
3. Memory Management
4. File Descriptor
5. Resource accounting (and limits)
6. Statistics (e.g. profiling info)
7. Signal actions
8. User Structure
BSD processes have the following states:

Nonexecuting processes in the srun or ssleep state are queued.
Windows NT is multithreaded, and allocates handles for managing processes and resources. The access token identifies the user, and their security permissions.

Figure 15: An NT Process and Its Resources
37 – Hardware Support for Debugging

Debugging requires being able to stop and start a process. Hardware support includes:

- A special *break point* interrupt to suspend the debugged process and signal the debugger.

- Hardware break points which can detect:
  - Data Reads
  - Data Writes
  - Instruction Reads/Execute

Many existing debuggers do not exploit hardware break points
Traditionally the \textit{ptrace} system call is used for debugging. The system call (in FreeBSD) looks like:

\begin{verbatim}
int ptrace(int request, /* Selects Debugging Action */
           pid_t pid,    /* Which process to debug */
           caddr_t addr, /* What address to access */
           int data)     /* What data to transfer */
\end{verbatim}

Which permits:

- Attaching to a process (i.e. debug a running process)
- Detaching a process (i.e. stop debugging a running process)
- Access memory associated with a process:
  - Read Data
  - Write Data
  - Break on Instruction Read/Execute
  - Get/Set Register Values
• Signal (interrupt) the debugged process.
• Notify the operating system that this process is being debugged. (using the PTRACE_ME flag.

Since some calls can return -1, a global variable (errno) needs to be checked to detect error conditions.
Traditionally the \textit{ptrace} system call is used for debugging. The system call (in FreeBSD) looks like:

\begin{verbatim}
int
ptrace(int request,    /* Selects Debugging Action */
    pid_t pid,       /* Which process to debug */
    caddr_t addr,    /* What address to access */
    int data)        /* What data to transfer */
\end{verbatim}
The Unix ptrace system call permits:

- Attaching to a process (i.e. debug a running process)
- Detaching a process (i.e. stop debugging a running process)
- Access memory associated with a process:
  - Read Data (function result)
  - Write Data
  - Break on Instruction Read/Execute
  - Get/Set Register Values
- Signal (interrupt) the debugged process.
- Notify the operating system that this process is being debugged. (using the PTRACE_ME flag.

Since some calls can return -1, a global variable (errno) needs to be checked to detect error conditions.
41 – Unix Procs file system

Since ptrace requires one system call per int of data, Modern Unix have procs which permits:

- Allows debugger to get more than an int worth of data per system call (vs. ptrace).
- Each current process has an entry in the directory:
  /proc
  by process id (pid).
- To access data in a process being debugged, the debugger:
  - Opens the appropriate procs file
  - Seek to the appropriate position in memory
  - Read memory (like reading from a file!)
  - Write memory (like writing to a file)
  - Close the procs file
References

