| | |
|---|---|
| Process Creation | • A process (sometimes called a task) can create another process<br>○ Parent process – the creating process<br>○ Child process – the process created<br>• Tree – parent and its children |
| | • Child process can be:<br>○ Duplicate of the parent process (same program/data)<br>○ New program loaded into it (using exec)<br>▪ Replaces process memory space with the new program<br>• How many children can you fork from a process?<br>○ Limited based on the amount of memory, maximum value for pid<br>○ On a 64-bit system could be ~4 million, but typically related to resources |
| Process Termination | • A process terminates when it finishes executing its final statement and asks the OS to delete it by using the exit() system call<br>• A return value can be sent to its parent via the wait() system call<br>• All system resources are deallocated |
| | • Parent typically is the only one that can terminate a child process<br>• Parent users from terminating other users' processes |
| | • Possible reasons for child termination:<br>○ Child has exceeded usage of resources allocated<br>○ Task assigned to the child is no longer required<br>▪ Multiple children processes -> one finishes<br>▪ Child process -> downloading file<br>○ The parent is exiting and the OS does not allow the child to continue if its parent is terminated<br>▪ Cascading termination |
| | • A process who has terminated, but the parent has not yet called wait() is called a **zombie process**<br>• Once the parent calls wait(), the process identifier and zombie process and its entry in the process table is released<br>• If the parent did not invoke wait and instead terminates, the child then becomes an **orphan** |
| Interprocess Communication | • Cooperating – any process that can be affected by another process. Any process that shares data with another process is a cooperating process<br>• Independent – any process that cannot affect or be affect by another process |
| | • Why cooperate?<br>○ Information sharing (e.g., shared file)<br>○ Computation speedup (e.g., run in parallel)<br>○ Modularity (e.g., system functions in separate processes)<br>○ Convenience (e.g., work on multiple tasks at the same time) |
| | • Two fundamental models for interprocess communication<br>○ Shared Memory |

| | |
|---|---|
| | ▪ Reading/writing data to a shared region of memory<br>○ Message passing<br>▪ Useful for exchanging smaller amounts of data |
| Shared-Memory Systems | • Producer/consumer: one process produces, the other process consumes<br>○ What are some examples of a Producer/Consumer scenario |
| | • Unbounded buffer – no practical limit on the size of the buffer<br>○ Producer can always produce new items<br>• Bounded buffer – fixed buffer size<br>○ Producer waits if buffer is full<br>○ Consumer waits if buffer is empty |
| Message-Passing Systems | • Symmetry – sender and receiver name each other to communicate<br>• Asymmetry – sender names the recipient but not  vice versa<br>○ Send(P, message) – send a message to P<br>○ Receive(id, message) – receive a message from any process, the variable id is set to the name of the process communicated with |
| Lab Questions<br>• Parent process runs<br>• Accepts input from user<br>• Forks a child process<br>• When the child process is forked, exec() is used to replace the child process with the command given by the user<br>○ Exec(ls, [argument passed to ls]) | |