AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

The Unix Process

Objectives

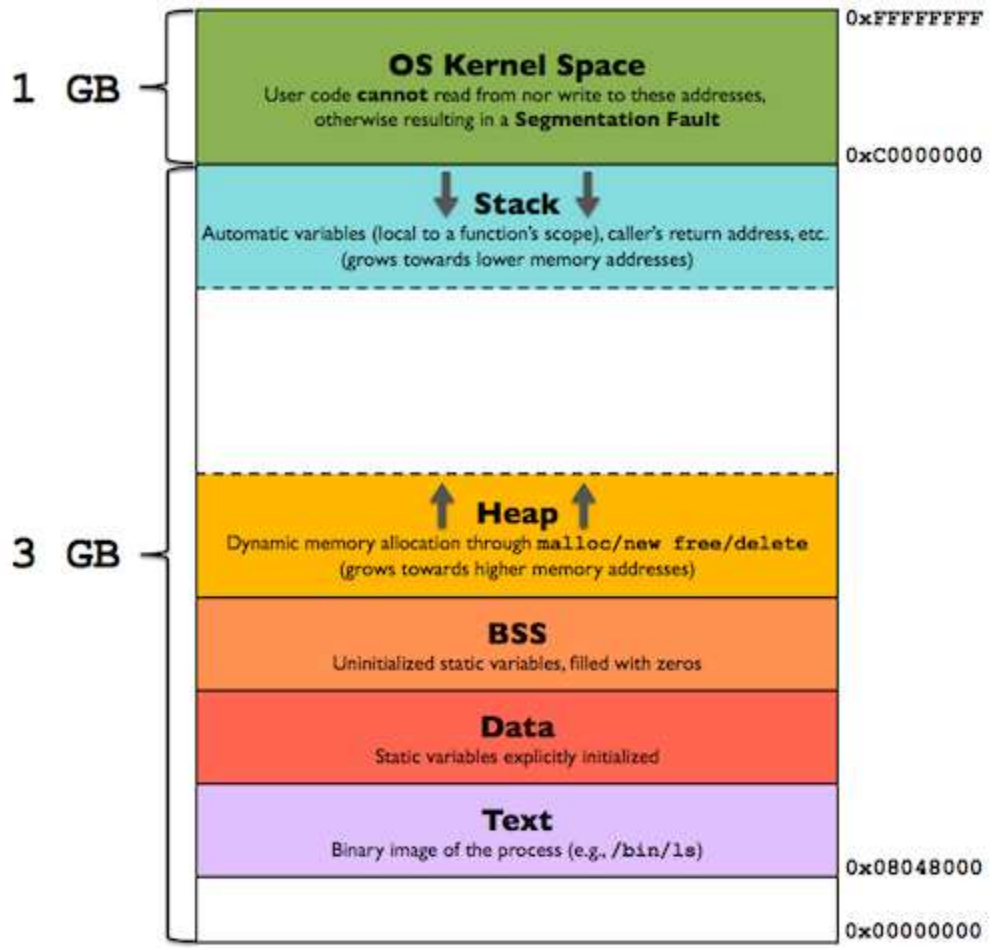
- To study the basic structure of Unix process
- Gives a preliminary view of process state transition diagram.

Unix Process - Introduction

- A processes can be defined as independent streams of computation which, as far as the kernel is concerned, compete over the machine's resources.
- Process in a Unix system is an entity created by fork() system call.
- The process which calls the fork() is the parent process and the new process is the child process.
- Every process will have a parent process and parent can have many child processes.
- Process 0 is handcrafted at boot time, which is the first process in Unix.

Memory of processes in Unix

- The memory allocated to UNIX processes can be divided into three logical parts:
  - Text: Machine code of the program
  - Data: contains representations of data pre-set to initial values. Also includes the amount of space to be allocated by the kernel for uninitialized data.
  - Stack: Usually contains procedure-based, downward growing, data frames.

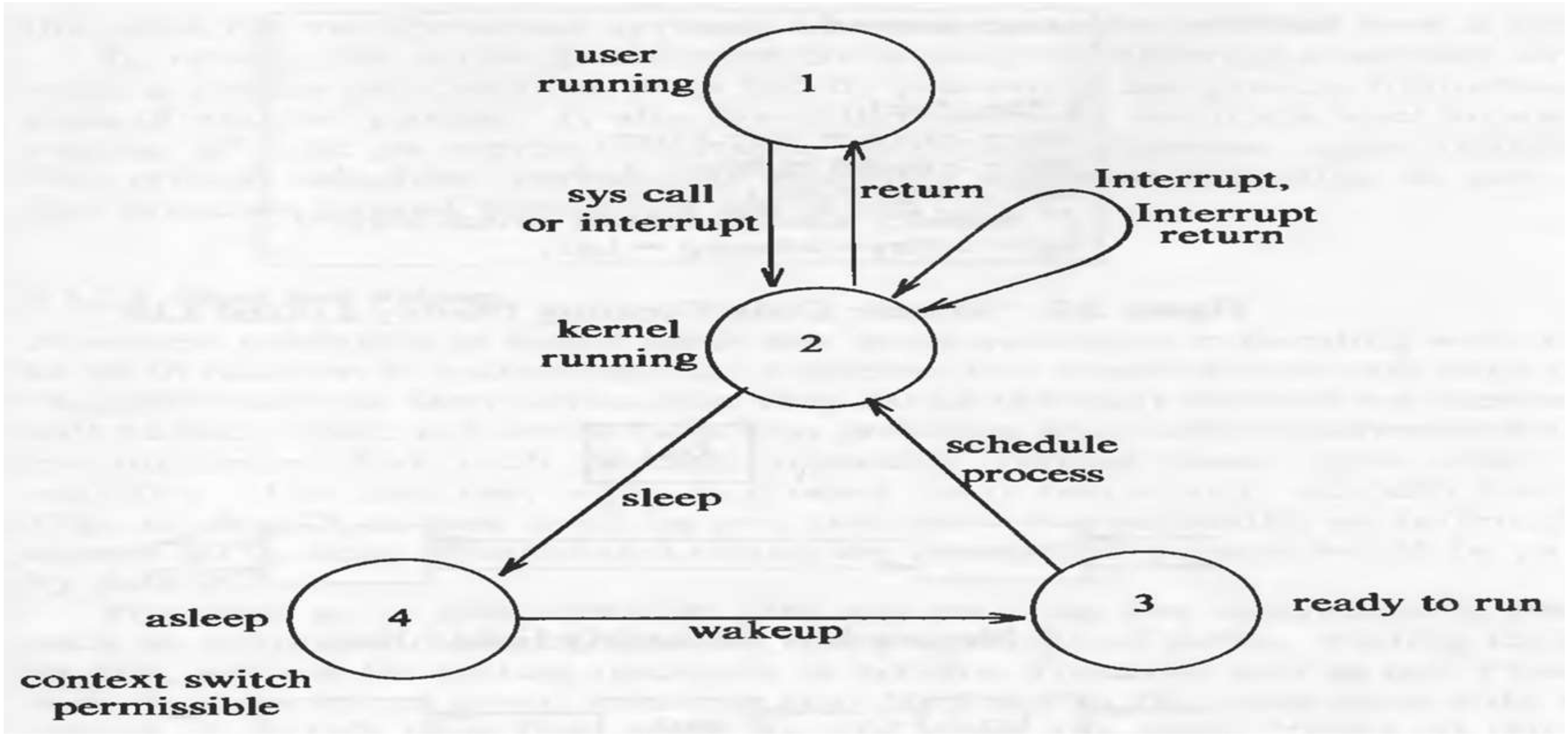


0xFFFFFFFF
1 GB
OS Kernel Space
User code cannot read from nor write to these addresses, otherwise resulting in a Segmentation Fault
0xC0000000
Stack
Automatic variables (local to a function's scope), caller's return address, etc. (grows towards lower memory addresses)
3 GB
Heap
Dynamic memory allocation through malloc/new free/delete (grows towards higher memory addresses)
BSS
Uninitialized static variables, filled with zeros
Data
Static variables explicitly initialized
Text
Binary image of the process (e.g., /bin/ls)
0x08048000
0x00000000

Modes of Execution

- Process in Unix can execute in two modes
  - User Mode
    - The user application are executed in user mode.
    - Not able to directly access hardware or reference memory
    - Can access these via System calls
  - Kernel Mode
    - The executing code has complete and unrestricted access to the underlying hardware.
    - Usually reserved for the lowest-level, most trusted functions of the operating system.
- Each one uses separate stack.

Process State Transition Diagram



Unix Process Management

- Process is an instance of program in execution.
- During the life time of a process, it uses several resources.
- Unix is a multi tasking, multi programming system, so a process in execution compete for system resources.
- Based on this a process at a specified instance of time can be in a particular state.
- The lifetime of a process can be conceptually divided into a set of states that describes the process.

Process States

- The complete set of process state is given below
  1. Running in user mode
  2. Running in Kernel mode
  3. Ready to run
  4. Sleeping in Memory
  5. Ready to run but swapped
  6. Sleeping in swap space.
  7. Pre-empted.
  8. Newly created.
  9. Zombie

Process States

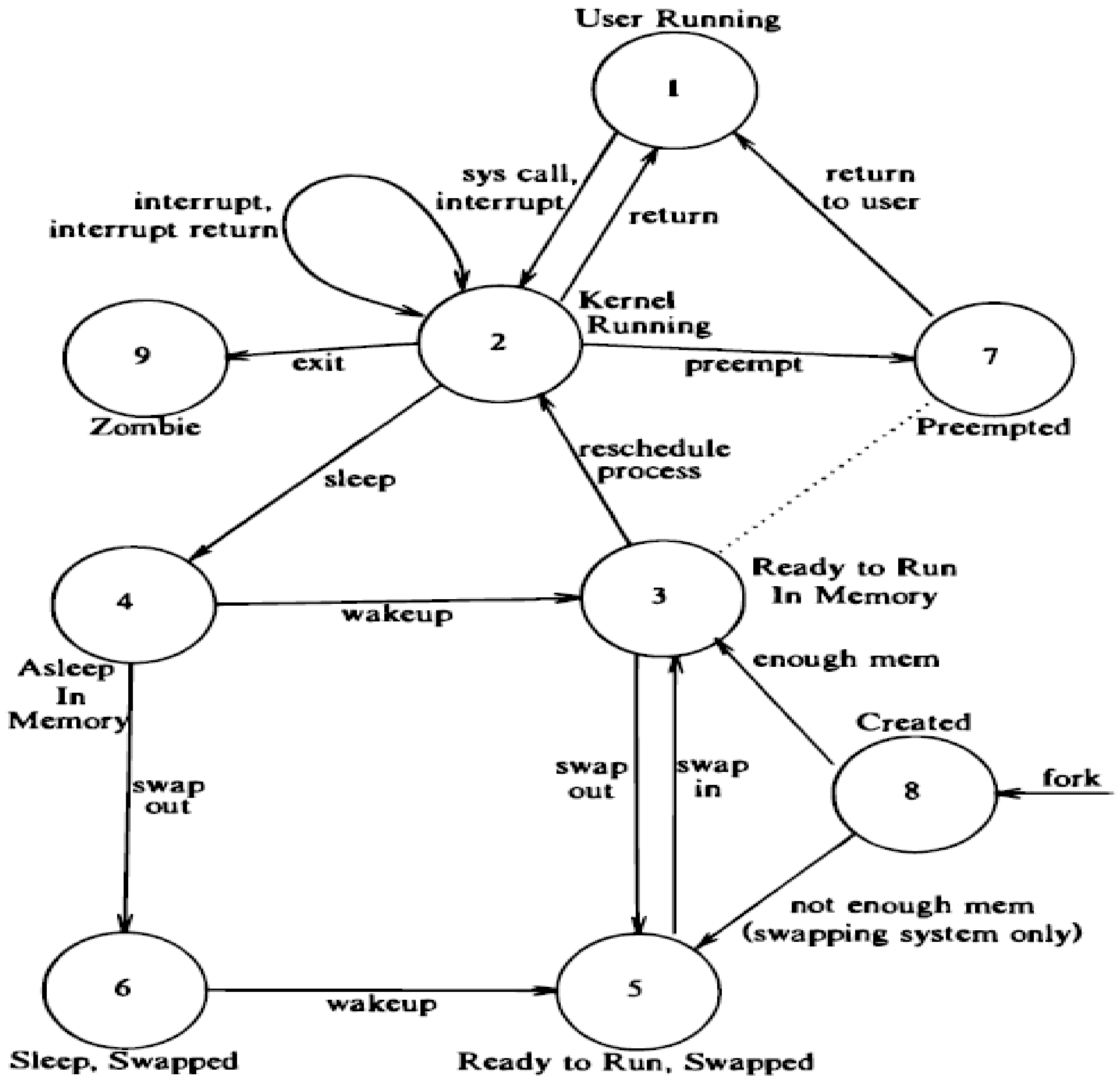


Figure 6.1. Process State Transition Diagram

Process State Transition

- The process enters the *created* state when the parent process executes the fork system call.
- If there is enough memory to accommodate this process then it moves to ready to run in memory state (3)
- If not enough memory it is moved to ready to run, and swapped out state (5). The process is ready to run but it is send to swap area, since no space is available in memory.
- From state 3, the scheduler will eventually pick the process and the process enters the state *kernel running (2)*, where it completes its part of *fork* system call.

Process State Transition

- After the completion of system call, it may move to *user running (1)*.
- When interrupts occur (such as system call), it again enters the state *kernel running (2)*.
- After the servicing of the interrupt the kernel may decide to schedule another process to execute, so the first process enters the state *pre-empted (7)*.
- The state *pre-empted (7)* is really same as the state *ready to run in memory*. Eventually, it will return to *user running* (1)again.

Process State Transition

- When a system call is executed, it leaves the state *user running* and enters the state *kernel running*.
- If in kernel mode, the process needs to sleep for some reason (such as waiting for I/O), it enters the state *asleep in memory (4)*.
- When the event on it which it has slept, happens, the interrupt handler awakens the process, and it enters the state *ready to run in memory (3)*.

Process State Transition

- When a process is in Ready to run and in memory, it may be swapped out to make room for another process. Then it goes to Ready to run , Swapped state.
- Eventually, swapper chooses the process as most eligible to run and it re-enters the state *ready to run in memory*.
- And then when it is scheduled, it will enter the state *kernel running*.
- When a process completes and invokes *exit* system call, thus entering the states *kernel running* and finally, the *zombie* state.
- A process that has completed execution but still has an entry in the process table.

Process State Transition

- When a process is asleep in memory (4) state and the I/O even is not happening, then it may be move out to swap area.

- The process enters to Sleep, Swapped (6) state.

- When the I/O is completed (wakeup), then it is in ready to run, swapped(5) state and when the space is available in memory it is bring into ready to run in memory (3) state.

Process State Transition

- The process transitions follow a rigid model encoded in the kernel, reacting to events in a predictable way according to formulated rules.

- For example, no process can pre-empt another process executing in the kernel.

Kernel Data Structures to Describe Process States

- Two kernel data structures describe the state of a process:
  - The process table entry
    - The process table contains information that should be accessible to the kernel.
  - u-area
    - u-area contains the information that should be accessible to the process only when its running.
- Kernel allocates space for u-area only when creating a process.

Process Table Entries

- The process table contains the information required by the kernel
  - Process State
  - Process IDs
  - User IDs
  - Pointer to text structure for shared text areas
  - Pointer to a page table for memory management
  - Scheduling parameters, including the "nice" value which determines priority
  - Timers for resource usage
  - A pointer to the process u-area

Process Table Entry

- State of the process
- Pointer to process and its u-area in main memory or in secondary storage.
- Size of the process : helps kernel to know how much space to allocate for the process.
- Several user identifiers (user IDs or PIDs) specify the relationship of processes to each other. These ID fields are set up when the process enters the state *created* in the *fork* system call.
- Event descriptor when the process is *sleep*ing.
- Scheduling parameters allow the kernel to determine the order in which processes move to the states *kernel running* and *user running*.
- Various timers give process execution time and kernel resource utilization. These are used for calculation of process scheduling priority. One field is a user-set timer used to send an alarm signal to a process.

U Area

- An operating system maintains a region called u-area. User area which holds the specific information of process and stored in a stack segment.
  - Privileges of the process – eg : file access rights
  - Timer Field : determines the time the process spent executing in user mode and in kernel mode.
  - The control terminal field identifies the "login terminal" associated with the process if one exists.
  - An error field records errors encountered during a system call.
  - A return value field contains the result of system calls.
  - I/O parameters describe the amount of data to transfer, the address of the source (or target) data array in user space, file offsets for I/O, and so on.
  - The current directory and current root describe the file system environment of the process.

Summary

- Discussed what is process and how a process is created in Unix
- The in memory representation of Unix process
- User mode and kernel mode of execution
- Process states and state transition diagram
- Kernel data structures to describe a process

Reference

- “The design of Unix Operating System”, Maurice J Bach