Processes

Dr Andrew Scott

a.scott@lancaster.ac.uk

1

What is a process?

- Running instance of a program
  - Program loader,
    invoked by exec( ), retrieves and unpacks executable
    - On disk formats, such as Executable and Linkable Format (ELF) files, include metadata giving expected memory layout
- Note a thread is a flow of execution within process
  - As most context/ state information per process …threads much more lightweight
    - Per thread state little more than CPU registers
  - Kernel may, or may not, natively support threads
    - Relatively easy to implement user space threads

2

Process Context

- Processes defined by their ***context***, including
  - Contents of CPU registers
  - Memory map/ layout (segments, pages, …)
  - Open files and current position in each
  - On-going communication state
    - Message queues
    - Signals e.g. KILL, semaphores, …
  - Configured timers or alarms
  - Accounting info. / Resource usage
    - Return state of any children

3

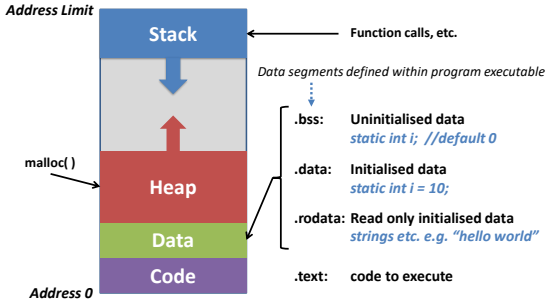
Basic Process Control Block (PCB)

- Holds management information/ context, one per process

Process Management	Memory Management	File Management
Processor Registers	Address of Text Segment	Root directory
Process state	Address of Data Segment	Working directory
Priority	Address of Stack Segment	Open file descriptors
Scheduling parameters	Address of Page Directory	Effective User ID
Process ID		Effective Group ID
Parent process		
Process group		
Signals		
Start time		
CPU time used		
Total children's CPU time		
Next alarm		

4

Program Address Space

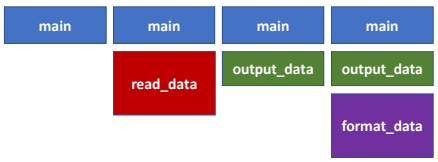


BSS (Block Started by Symbol) is an historic name used by a 1950's assembler for the IBM 704

5

Process Stack

- Common way of implementing function calls
  - Functions parameters + return address placed on stack before call
  - There is no standard, could use mix of registers and stack, …
- Grows and shrinks in blocks (frames) with function calls
  - On x86 stack grows down, from high to low memory address



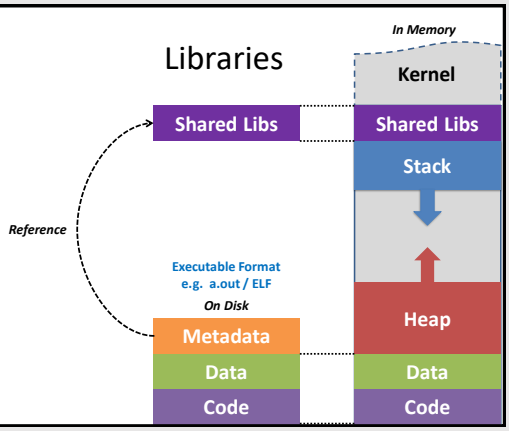
6

Process Heap

- Predefined memory areas often too restrictive
  - For example, arrays can't expand if more data than expected
- So, typically allocate memory dynamically as needed
  - Elements in linked-list, queue, graph, …
  - New object instances, etc.
- Use library calls to expand and (possibly) contract heap
  - memory = malloc( )
  - free(memory)

7

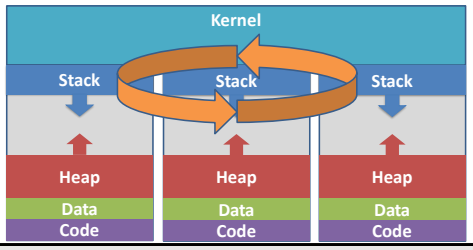
Libraries



8

Multiple Processes

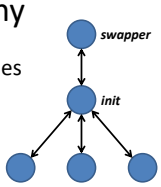
- Each process has running state (*context*)
  - Instruction and Stack pointers, status flags, …



9

Process Hierarchy

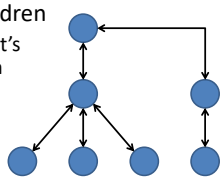
- Processes can *spawn* child processes
  - Limit on number imposed by system
    - Per process (for fairness)
    - System wide (for system stability)
- Each child has parent process
  - Ultimately responsible for resources used by child
- In Unix, */sbin/init* (process 1) is ultimate parent of all processes



10

Resource Allocation

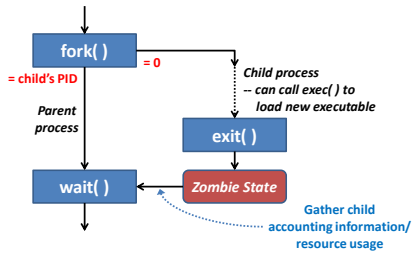
- Should take account of process hierarchy
- Parents responsible for children
  - Children take share of parent's time and resource allocation
  - Shouldn't be able to spawn lots of child processes to obtain more resources



11

Creating a Child Process

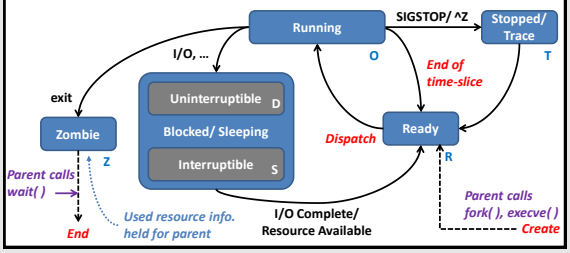
- In Unix this carried out by *fork( )* system call
  - Returns child's Process ID to parent and 0 to child



12

Typical Process States

- Interruptible: operation can be stopped
- Uninterruptible: operation must complete



13