ECE 332 – Embedded Systems Laboratory

Big Picture Lab 4

Prof. Sandip Kundu

Acknowledgement

- These slides are a result of cumulative contributions from Prof. Burleson, Koren, Kundu and Moritz.

- Materials have also been adopted from the textbook: Wolf, Computers as Components, Morgan Kaufman, 2005

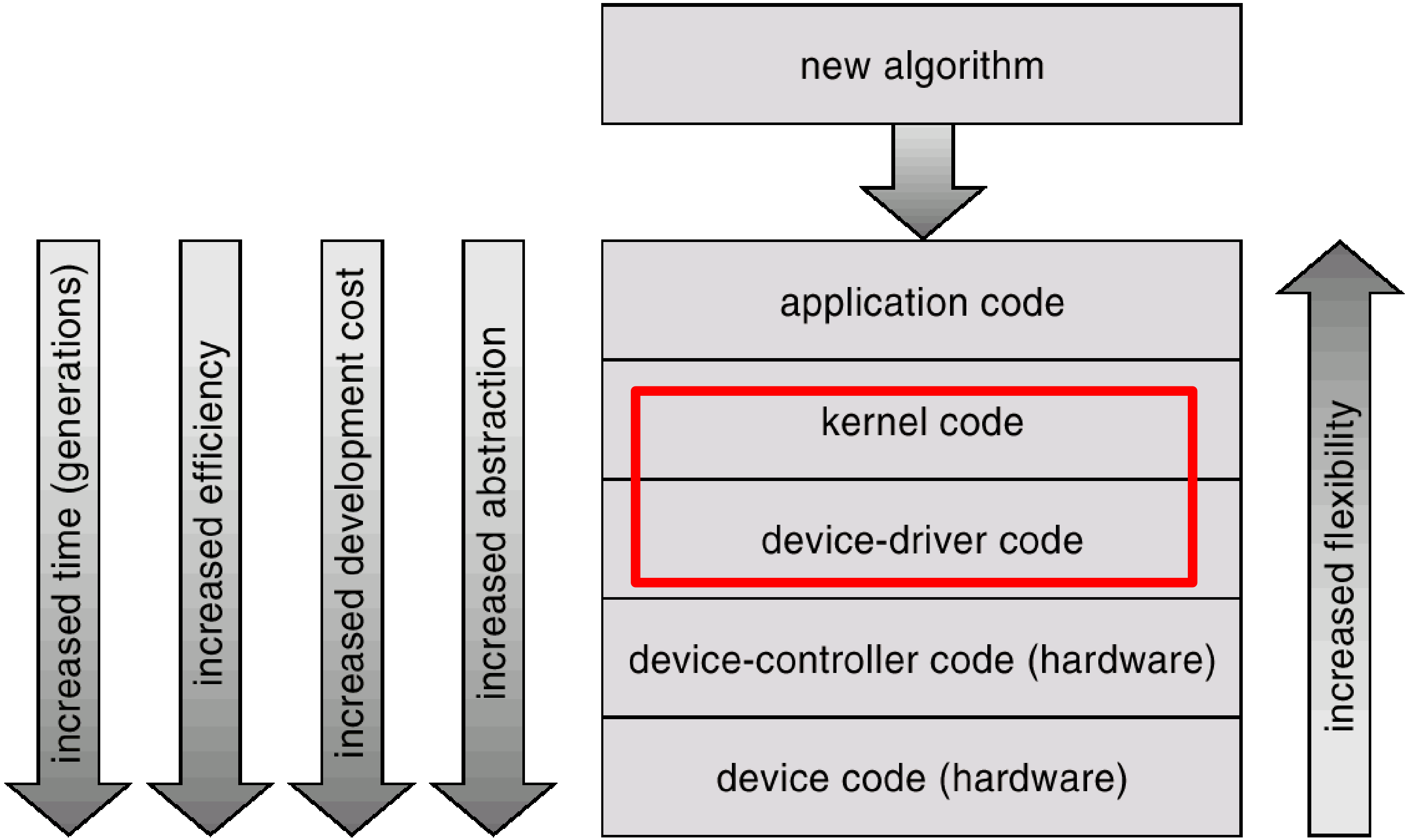
Readings

- Wolf, Computers as Components, Chapter 6 pp. 293-352
- Multiple Tasks and Multiple Processes
- Pre-emptive Real-time Operating Systems
- Priority-Based Scheduling
- Interprocess Communication Mechanisms
- Evaluating OS Performance
- Power Management and Optimization for Processes
- Design Example Telephone Answering Machine

Operating system is just another program.. a big one

- The operating system controls resources:
  - who gets the CPU;
  - when I/O takes place;
  - how much memory is allocated.
- The most important resource is the CPU itself.
  - CPU access controlled by the scheduler of processes.
- OS needs to keep track of:
  - process priorities;
  - scheduling state;
  - process activation record.

Where is the OS?



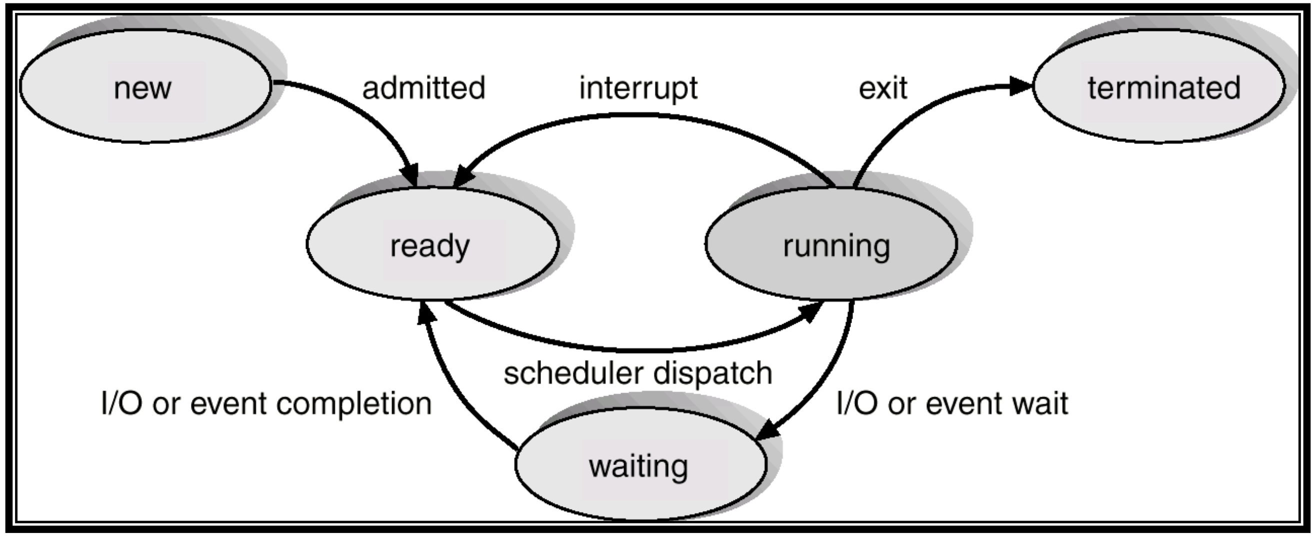
Process Concept

- Process = a program in execution (main thing to manage)
  - process execution must progress in sequential fashion.
- Processes may be created:
  - statically before system starts;
  - dynamically during execution.

Process State

- As a process executes, it changes state
- Each process may be in one of the following states (names vary across various OS):
  - new: The process is being created.
  - running: Instructions are being executed.
  - waiting: The process is waiting for some event to occur.
  - ready: The process is waiting to be assigned to a processor.
  - terminated: The process has finished execution.

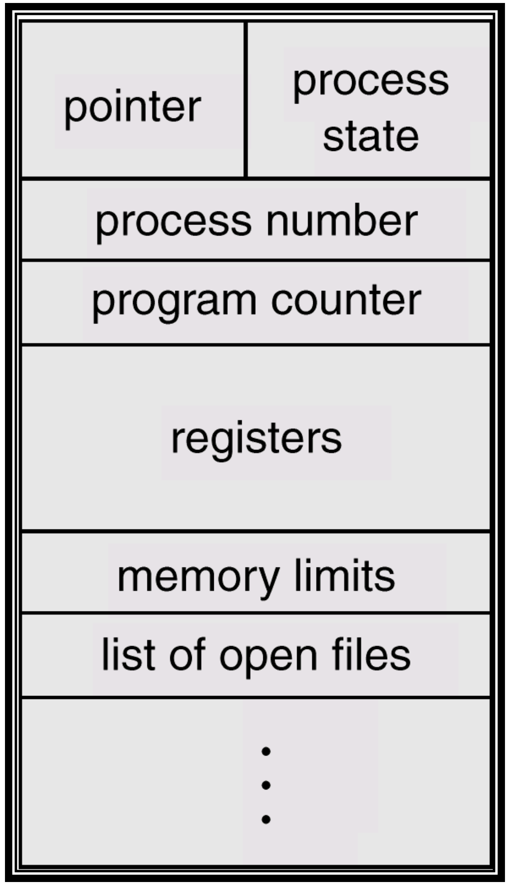
Diagram of Process State in the OS



Process Control Block (PCB)

- Information associated with each process in the OS, i.e., the process related data structure.
- The PCB contains:
  - Process state (running, waiting, …)
  - Program counter (value of PC)
  - Stack pointer, General purpose CPU registers
  - CPU scheduling information (e.g., priority)
  - Memory-management information
  - Username of owner
  - I/O status information
  - Pointer to state queues, ..

Process Control Block (PCB)



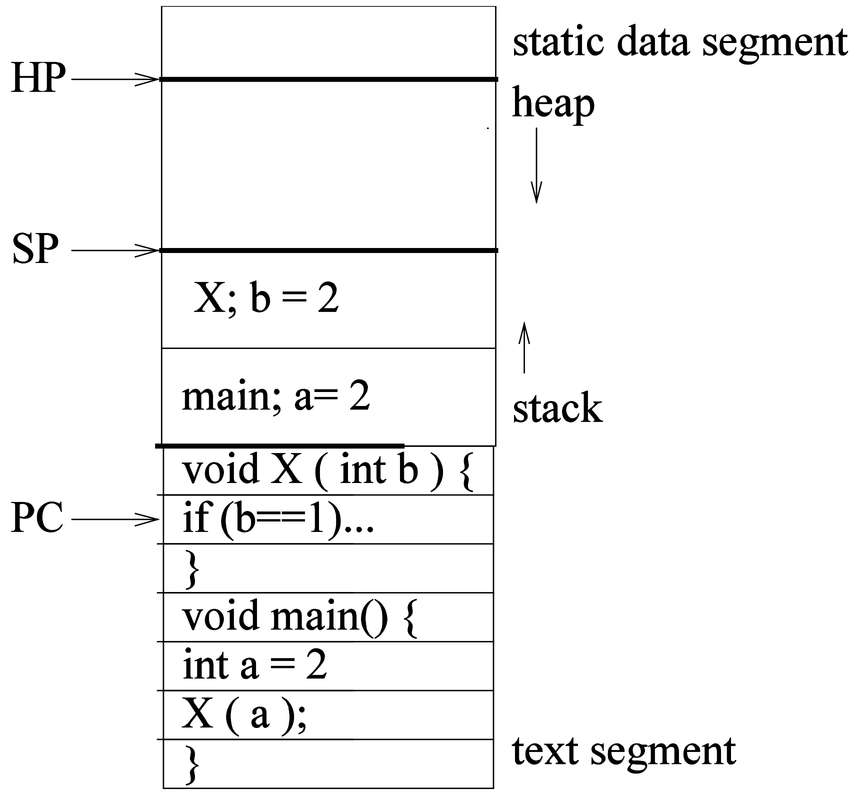
Example Process State in Memory

What you wrote:

PC->

```
void X(int b){
        If (b==1) ..
}

main(){
        int a = 2;
        X(a);
}
```

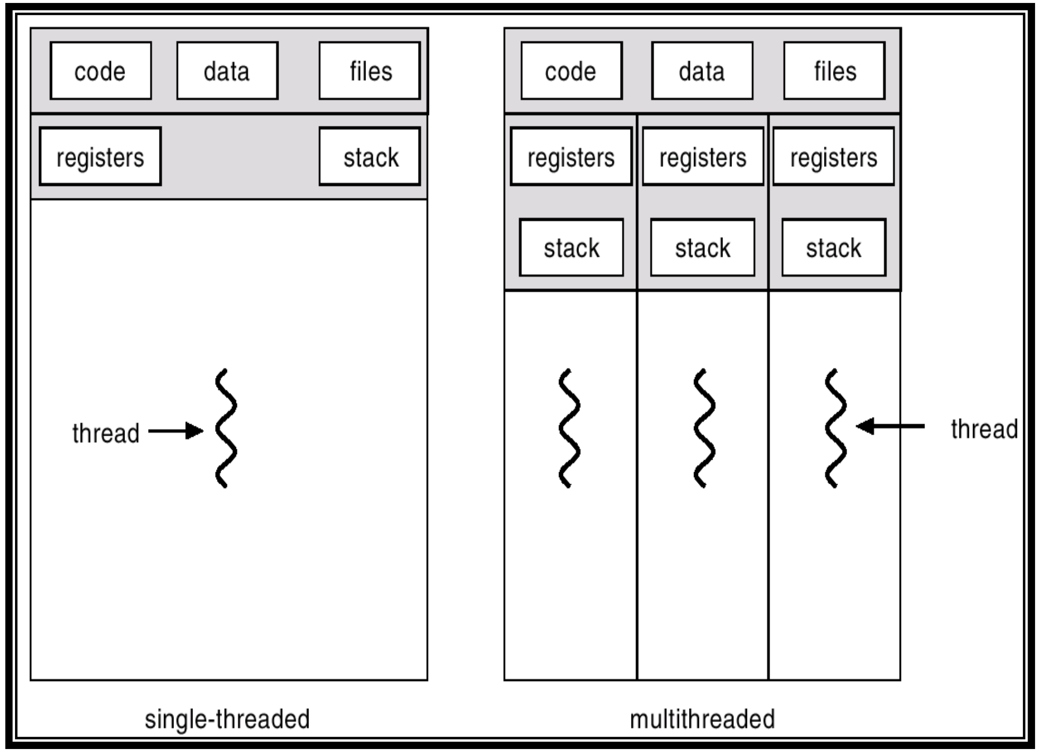
In memory:



Process State

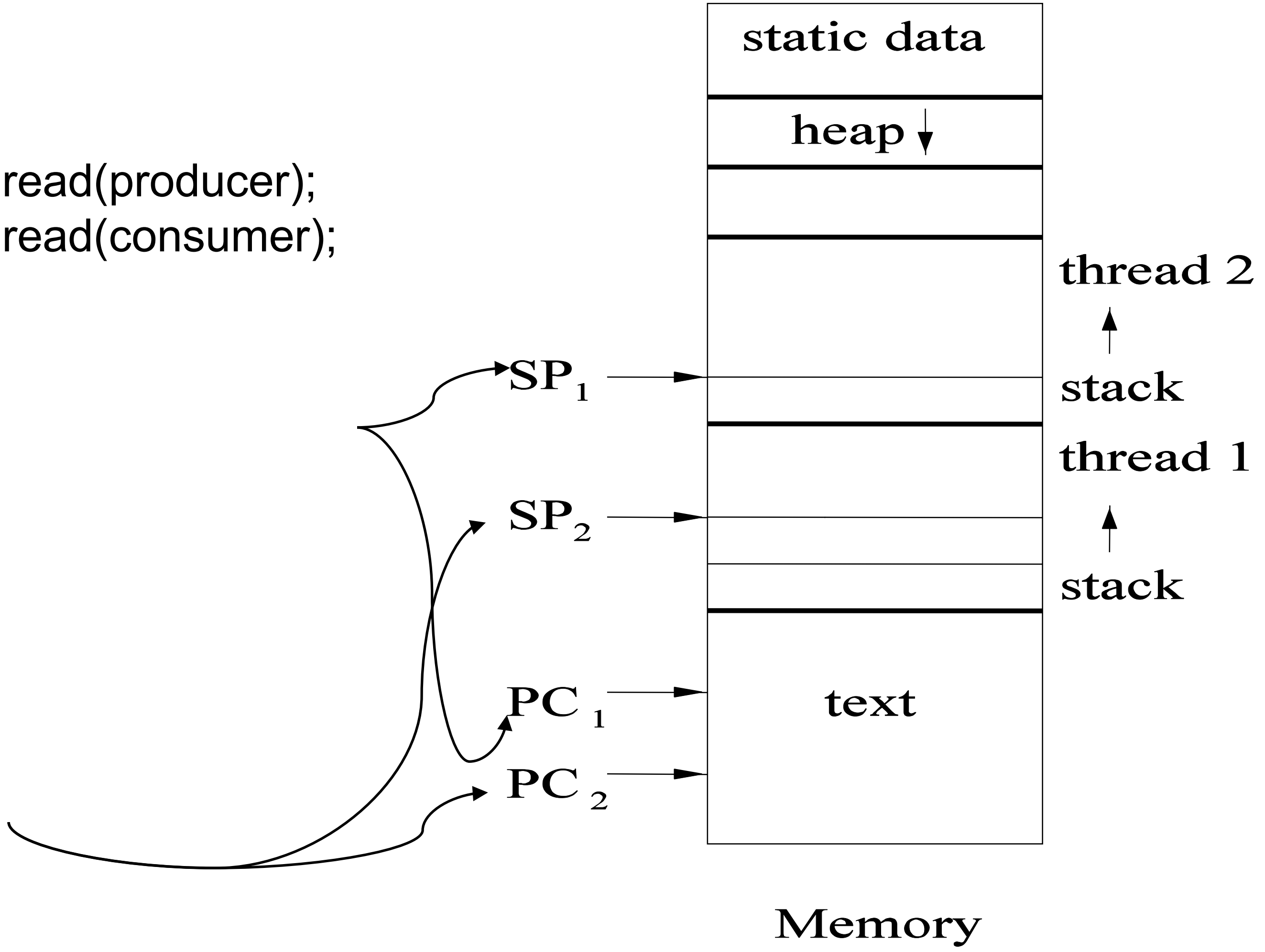
Single and Multithreaded Processes

- A thread = stream of execution
- Benefits?



Example of Memory Layout with Threads

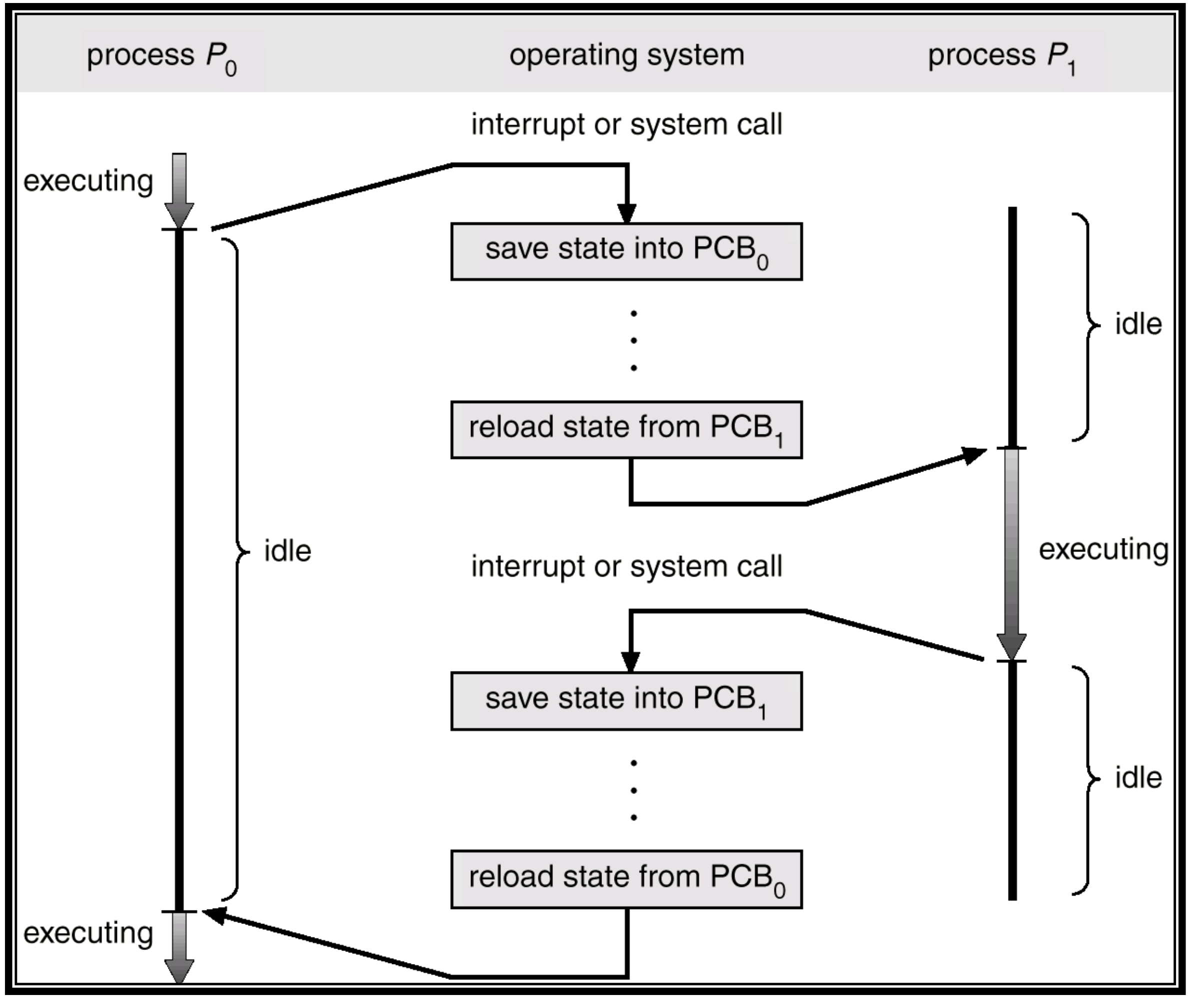
```
main() {
        ……
        fork_thread(producer);
        fork_thread(consumer);
        ……
}

producer() {
        ……
}

consumer(){
        …….
}
```



Embedded vs. general-purpose scheduling

- Workstations try to avoid starving processes of CPU access.
  - Fairness = access to CPU.
- Embedded systems must meet deadlines.
  - Low-priority processes may not run for a long time.
- Priority Driven scheduling
  - Each process has a priority.
  - CPU goes to highest-priority process that is ready.

CPU Switch From Process to Process



Interprocess Communication

- Interprocess communication (IPC): OS provides mechanisms so that processes can pass data.

IPC Styles

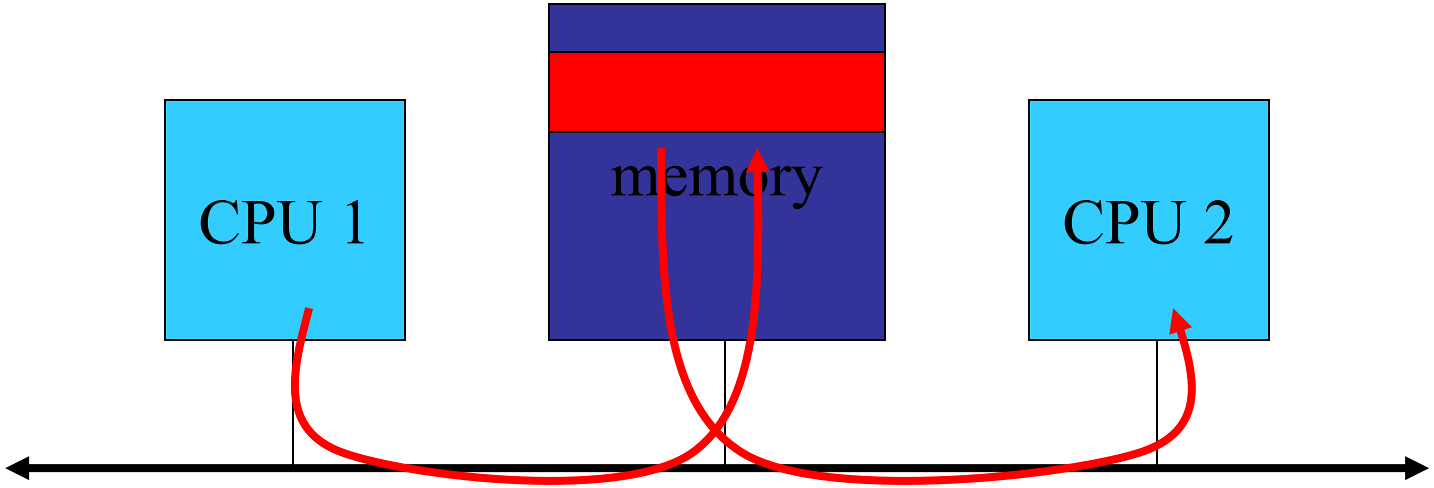
- Shared memory:
  - processes have some memory in common;
  - must cooperate to avoid destroying/missing messages.
- Message passing:
  - processes send messages along a communication channel---no common address space.

Critical Regions

- Critical region: section of code that cannot be interrupted by another process.
- Examples:
  - writing shared memory;
  - accessing I/O device.

Shared Memory

- Shared memory on a bus:



Race Condition in Shared Memory

- Problem when two CPUs try to write the same location:
  - CPU 1 reads flag and sees 0.
  - CPU 2 reads flag and sees 0.
  - CPU 1 sets flag to one and writes location with 123.
  - CPU 2 sets flag to one and overwrites location with 456.

  - CPU 1 thinks value is 123 since it checked flag but it is 456!

Synchronization Hardware – ISA Support

- E.g.,: Test and modify the content of a word atomically
  - Below pseudo-code for the hardware would implement in ISA.

```
boolean TestAndSet(boolean &target) {
        boolean rv = target;
        target = true;

        return rv;
}
```

Mutual Exclusion Lock with Test-and-Set

- Can be used to implement a simple lock
- Shared data:

```
boolean lock = false;
```

Process Pi

```
do {
    while (TestAndSet(lock)) ;
        critical section
    lock = false;
        remainder section
}
```

Wait here/test until/if Lock is TRUE, If it is not, set it and continue

Semaphores

- Semaphore: OS primitive for controlling access to critical regions.
  - Based on test-and-set or swap at implementation level
    - Binary semaphors similar to mutex locks shown earlier conceptually
    - Counting semaphors allow some # of players access to critical section
- Protocol:
  - Get access to semaphore.
  - Perform critical region operations.
  - Release semaphore.

BACKUP

What do you need to design a real embedded system?

- You know how to interface to simple switches and lights
- You know how to use memory of different types (SRAM, DRAM, EEPROM, Flash)
- You know how to interface to serial I/O (UART, USB, SPI, GPIO)
- You know how to interface to Ethernet/Internet
- You know how to write programs and process data in C
- What’s next?
  - Running multiple programs
  - Real-time?
  - Reliability? Security? Upgradeability?
- You need an Operating System!

Atomic test-and-set in ARM ISA

- ARM test-and-set provided by SWP: single bus operation reads memory location, tests it, writes it.
- Example mutex lock implementation

```
    EXPORT lock_mutex_swp
lock_mutex_swp PROC
    LDR r2, =locked
    SWP r1, r2, [r0]       ; Swap R2 with location [R0], [R0] value placed in R1
    CMP r1, r2             ; Check if memory value was 'locked'
    BEQ lock_mutex_swp     ; If so, retry immediately
    BX  lr                 ; If not, lock successful, return
    ENDP

    EXPORT unlock_mutex_swp
unlock_mutex_swp
    LDR r1, =unlocked
    STR r1, [r0]           ; Write value 'unlocked' to location [R0]
    BX  lr
    ENDP
```