Lecture Notes on Nov/30

A. Finite-State Machine
B. Inter-Process Communication – Part 1

ECE217 Data Structure and Algorithms

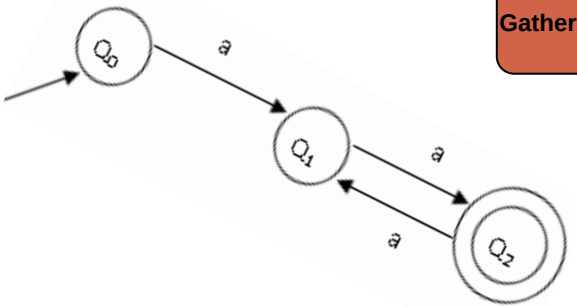
Instructor: Dr. Shayan (Sean) Taheri

Finite-State Machine (FSM)

- A finite-state machine/automaton comprises the following elements:
  - A sequence of input symbols (the input "tape").
  - The current location in the input, which indicates the current input symbol (the read "head").
  - The current state of the machine (e.g., denoted q_0, q_1, …, q_n).
  - A transition function which inputs the current state and the current input, and outputs a new (next) state.
  - One or more states may be marked as final states, such that the computation is considered successful if and only if computation ends in a final state.
  - An FSA can be represented graphically as a directed graph, where the nodes in the graph denote states and the edges in the graph denote transitions.
  - Final states are usually denoted by a double circle.

Case 1 (Not Accepted) :

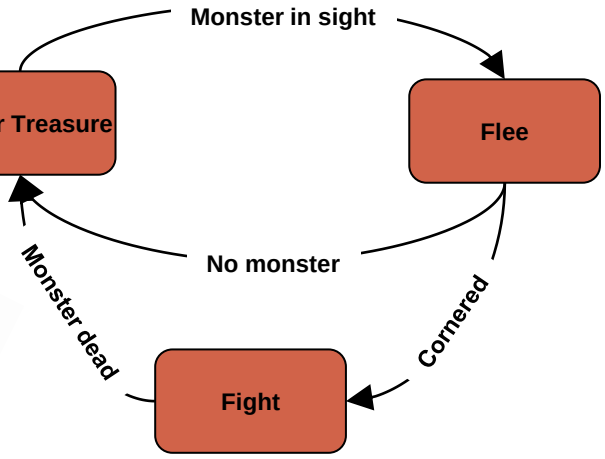
- ✓ **Input Sequence: aaa**
- ✓ **State Transitions: q0, q1, q2, q1**



Case 2 (Accepted) :

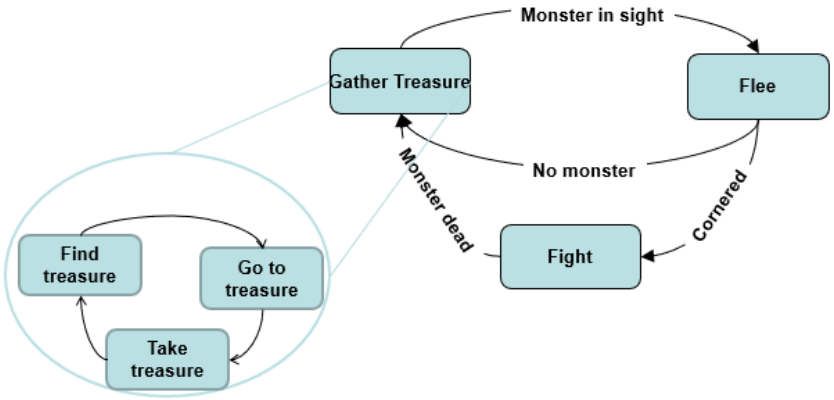
Input Sequence: aaaa

State Transitions: q0, q1, q2, q1, q2

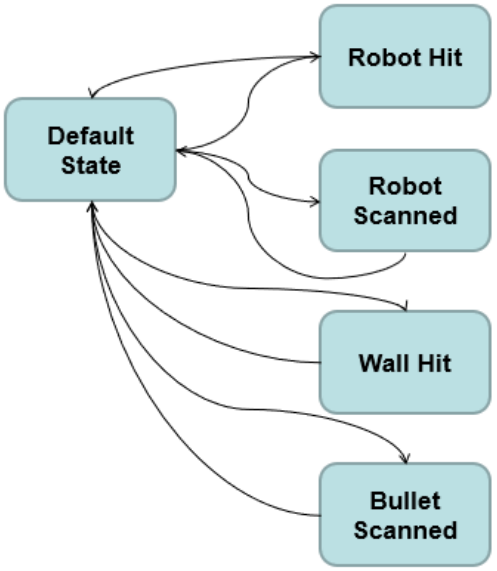


Finite-State Machine (Cont.)

- **Types**:
  - Hierarchical FSM (HFSM)
    - ✓Each state consists of sub-states.
    - ✓Better modularity.
  - Stack FSM (SFSM)
    - ✓It provides additional memory.
    - ✓Can be used to remember state history (push).
    - ✓Can return to previous state (pop).
    - ✓Enter a new state entirely and forget about the old one (replace).
  - Message-Passing FSM (MPFSM)
    - ✓Event-driven.
    - ✓Integration with other FSMs or game engines.
    - ✓Messages are in enumerated type.
    - ✓Used to notify of external changes of the world.



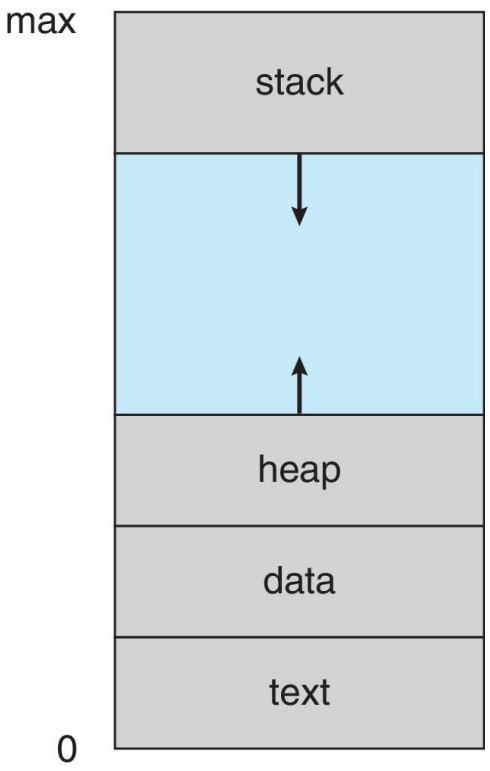
- **HFSM (Top-Right)**
- **MPFSM (Bottom-Right)**

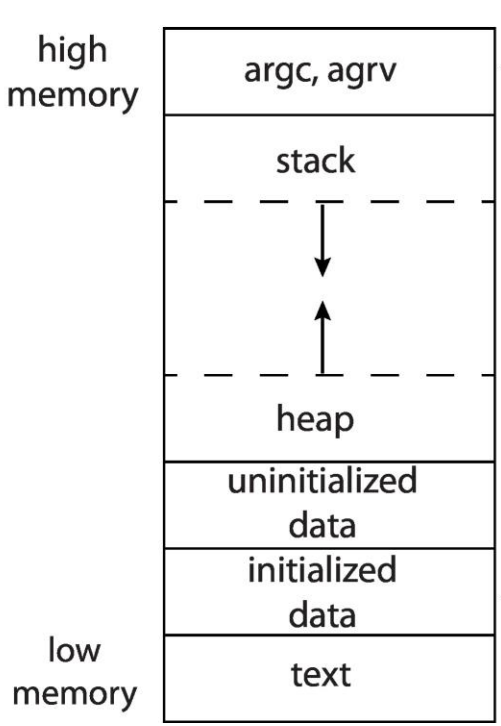


Concept of Process

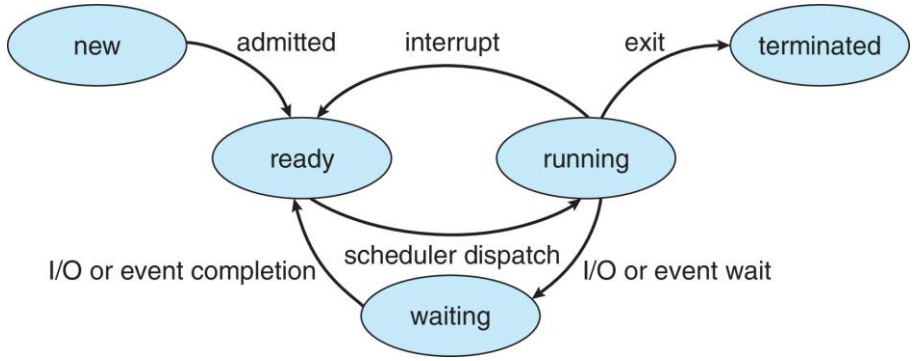
- An operating system executes a variety of programs that run as a process.
- **Process**: a program in execution; process execution must progress in sequential fashion.
- **Multiple Parts**
  - The program code, also called **text section**.
  - Current activity including **program counter**, processor registers.
  - **Stack** containing temporary data:
    - ✓Function parameters, return addresses, local variables.
  - **Data section** containing global variables.
  - **Heap** containing memory dynamically allocated during run time.
- Program is a ***passive*** entity stored on disk (**executable file**), but process is ***active***.
  - Program becomes process when executable file loaded into memory.
- One program can be several processes:
  - Consider multiple users executing the same program.

Concept of Process (Cont.)





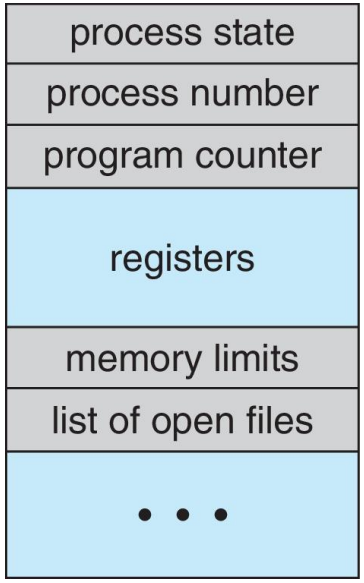

```
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```



- As a process executes, it changes **state**:
  - **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

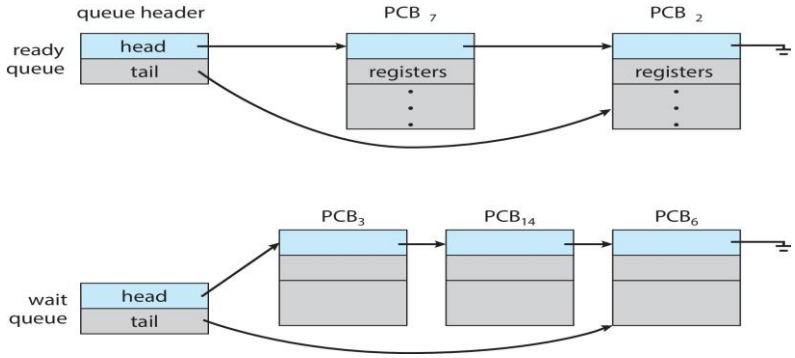
- **Process in Memory (Top-Left)**
- **Memory Layout (Top-Right)**
- **Process State (Bottom-Right)**

Concept of Process (Cont.)

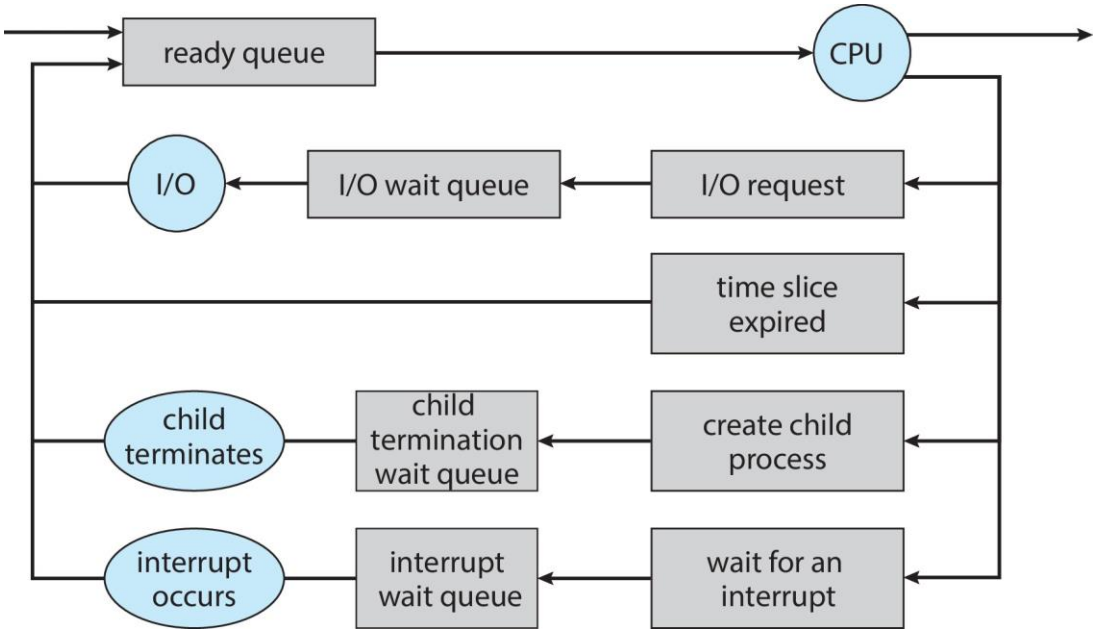


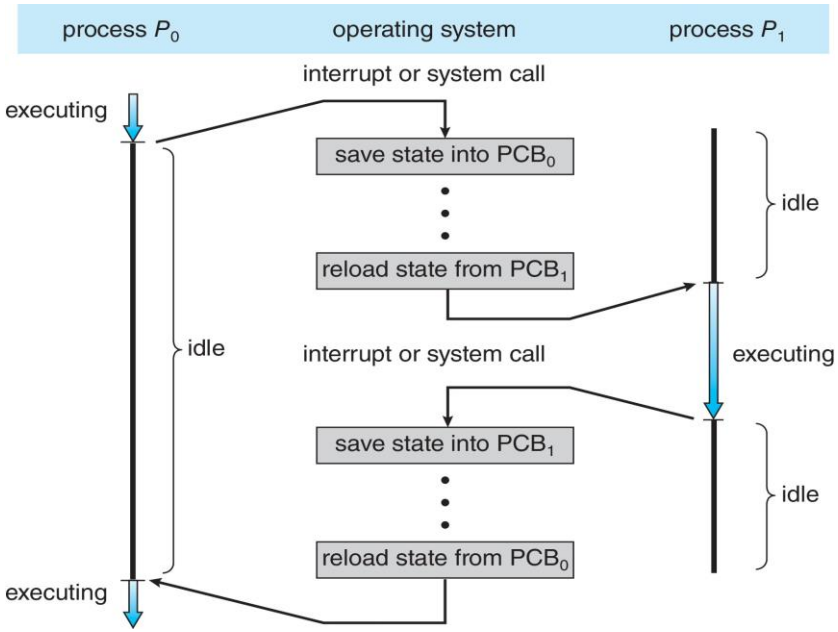
- **Process/Task Control Block (PCB)**:
  - Containing information associated with each process.
  - Process State: Running, waiting, etc.
  - Program Counter: Location of instruction to next execute.
  - CPU Registers: Contents of all process-centric registers.
  - CPU Scheduling Information with Priorities (e.g., scheduling queue pointers).
  - Memory-Management Information: Memory allocated to the process.
  - Accounting Information: CPU used, clock time elapsed since start, time limits, etc.
  - I/O Status Information: I/O devices allocated to process, list of open files, etc.
- **Thread**:
  - Consider having multiple program counters per process.
  - Then, multiple locations can execute at once → Multiple Threads of Control.
  - Best usage for multi-core or multi-CPU systems.
  - Having storage for thread details (i.e., multiple program counters in PCB.

Process Scheduling



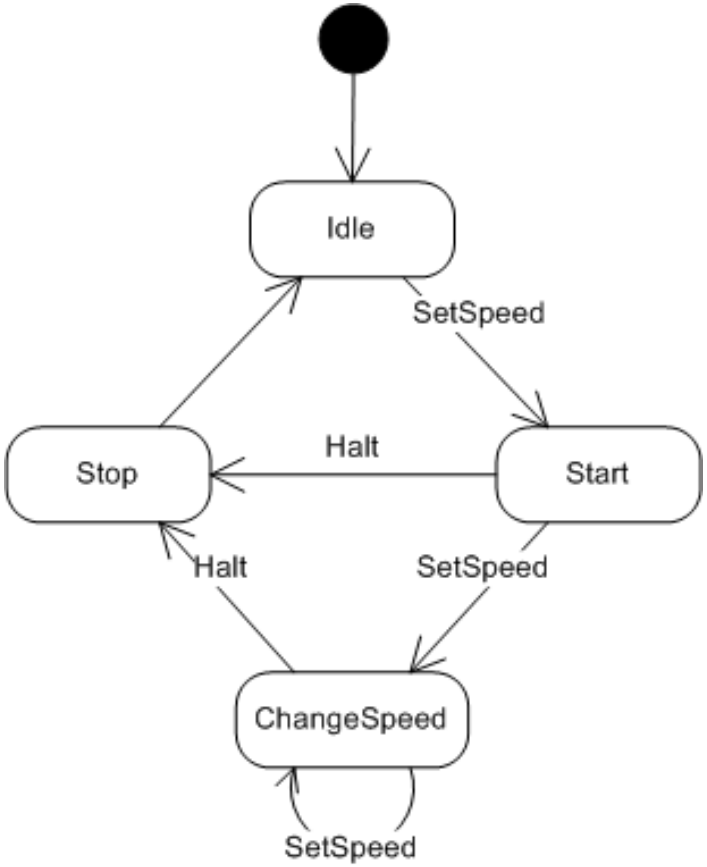
- **Maximize CPU Usage**.
- Quickly switch processes onto CPU core.
- Process scheduler selects among available processes for next execution on CPU.
- Maintains scheduling queues of processes:
  - **Ready Queue**: Set of all processes residing in main memory, ready and waiting to execute.
  - **Wait Queues**: Set of processes waiting for an event (i.e., I/O).
  - Processes migrate among the various queues.
- A **Context Switch** occurs when the CPU switches from one process to another.





Finite-State Machine in C++ Language

```
119  // Driver Code
120  int main()
121  {
122      // Task 1: Create an Engine in Your Finite-State Machine
123
124      // Task 2: Show the Machine Current State
125
126      // Task 3: Perform FSM Actions
127
128      return 0;
129  }
```



Assignment

- **Reading Assignment:**
  - Finite-State Machine in Wikipedia.
  - Process (Computing) in Wikipedia.
  - Thread (Computing) in Wikipedia.
  - Scheduling (Computing) in Wikipedia.

Questions?