

The background of the slide is a photograph of a night sky filled with stars. In the foreground, there's a silhouette of a person standing next to a yellow tent on a rocky hillside. The person is pointing upwards towards the stars. The overall atmosphere is serene and suggests a connection between the natural world and the concepts being discussed.

arm

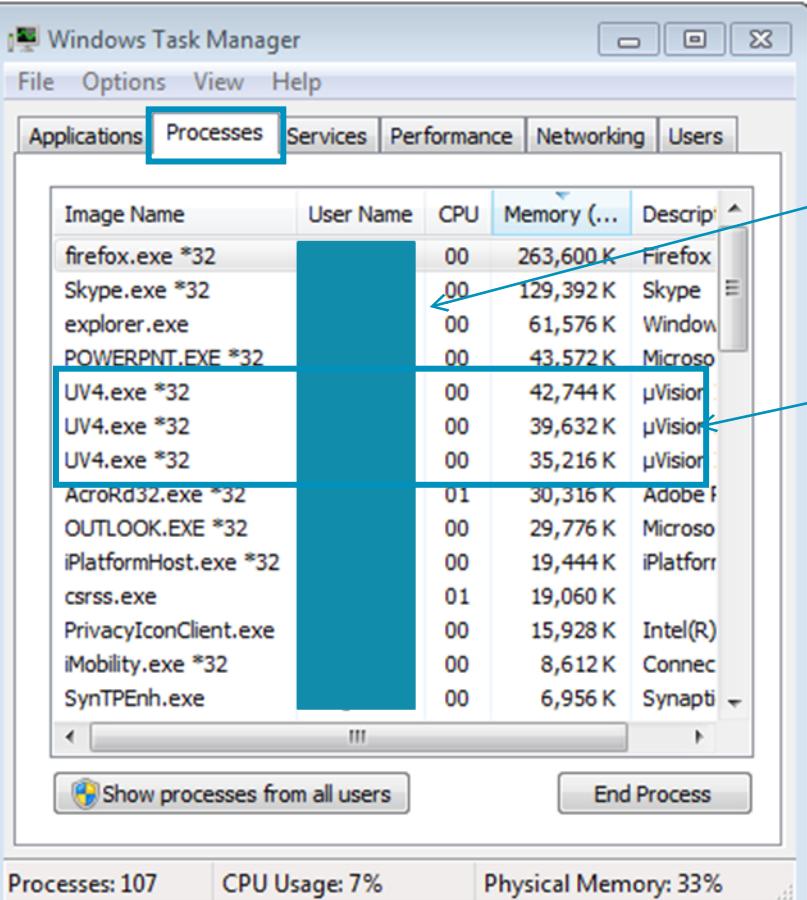
Processes

Operating System Lab-in-a-Box

Lecture Contents

- Process- What is it
- Memory lay out
- Switching between Processes
- RTX and Linux Examples
- Process Termination
- Context Switching and States

An Instance of a Running Program

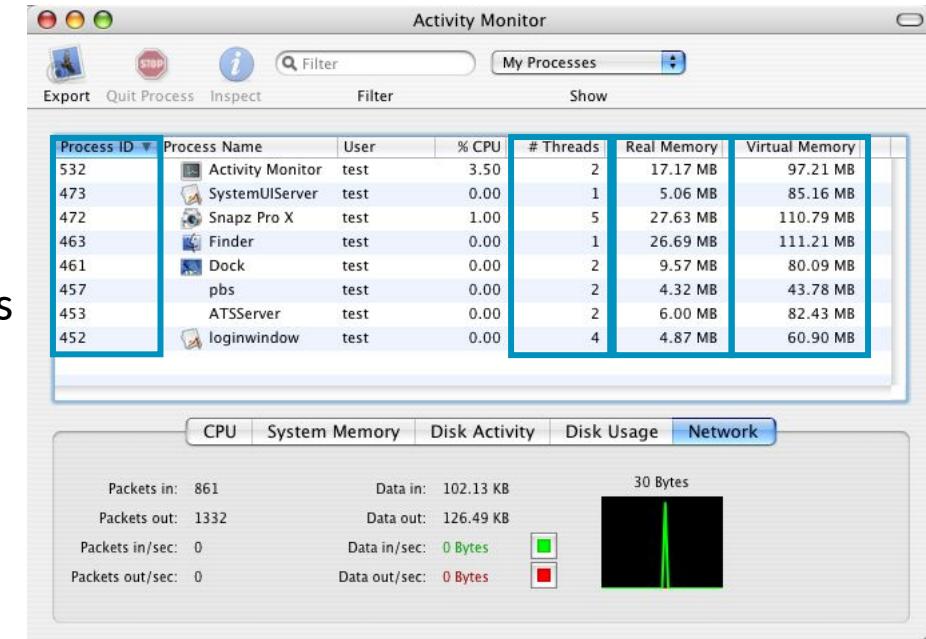


The Windows Task Manager is displayed with the 'Processes' tab selected. A blue box highlights the 'Processes' tab in the menu bar. The main table lists various processes, with three instances of 'UV4.exe *32' highlighted by a blue box. Arrows point from the text 'Your user name or SYSTEM/LOCAL SERVICE/NETWORK SERVICE' and 'Multiple instances of the μVision4' to the highlighted rows in the task manager.

Image Name	User Name	CPU	Memory (...)	Description
firefox.exe *32		00	263,600 K	Firefox
Skype.exe *32		00	129,392 K	Skype
explorer.exe		00	61,576 K	Window
POWERPNT.EXE *32		00	43,572 K	Microso
UV4.exe *32		00	42,744 K	μVision
UV4.exe *32		00	39,632 K	μVision
UV4.exe *32		00	35,216 K	μVision
AcroRd32.exe *32		01	30,316 K	Adobe F
OUTLOOK.EXE *32		00	29,776 K	Microso
iPlatformHost.exe *32		00	19,444 K	iPlatfor
csrss.exe		01	19,060 K	
PrivacyIconClient.exe		00	15,928 K	Intel(R)
iMobility.exe *32		00	8,612 K	Connec
SynTPEnh.exe		00	6,956 K	Synapti

Your user name or SYSTEM/LOCAL
SERVICE/NETWORK SERVICE

Multiple instances of the μVision4
Independent memory for each process
“Memory(Private Working Set)”



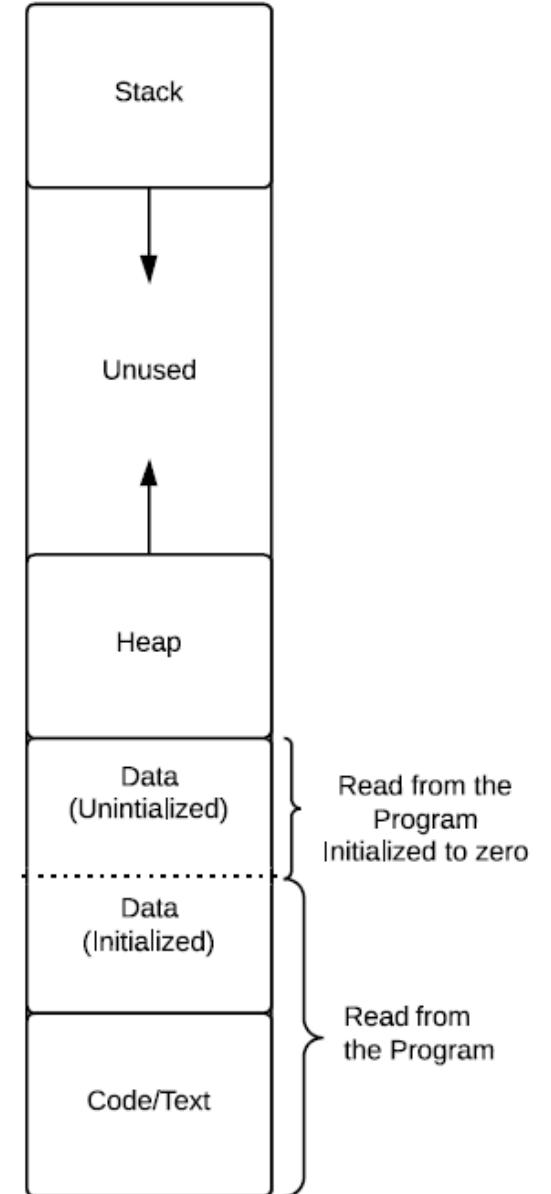
Try to explore the task manager, activity monitor, or similar utility on your favourite OS

An Instance of a Running Program

- A process can have
 - A CPU time allocation (virtual CPU, the role of OS)
 - Memory (real or virtual)
 - Process ID
 - Threads
- Are they aware of the existence of other processes?
 - The operating system's role is to create an “illusion” that a process has all it needs to be executed
 - An abstraction of hardware resources
 - Each process sees one dedicated processor and one segment of memory (although they are often shared with others)
 - But they can be aware of each other – inter-process communication (IPC)

Memory Layout of an Executing Program

- Code or Text
 - Binary instructions to be executed
 - A clone of the program
 - Usually read-only
 - Program counter (PC), points to the next instruction
- Static Data
 - Global/Constant/Static variables - shared between threads
 - If not initialized by program, will be zero or null pointer
- Heap
 - malloc/free
- Stack
 - Used for procedure calls and return
 - Stack Pointer(SP)
 - FILO (First In, Last Out)



Process – The Abstraction

- Switching between executions means the operating system has to keep track of all the execution context
- Includes:
 - Memory State (code, data, heap and stack)
 - CPU state (PC, SP and other registers)
 - Also the OS state
- Hence the abstraction of the process
- Program usually refers to the instructions that are stored on disk
- Process is the program with execution context
- Some OSs may use the term “task”, particularly in an embedded system context. We will use both terms interchangeably for this course
- Thread: a lightweight process; a process may have multiple threads which share the same system resources - faster creation, termination, switching and communication

Process Control Block

- The Process Control Block (PCB) or Task Control Block (TCB) maintains all the relevant information for the process:
 - Process ID
 - Process state
 - PC, SP and other registers (stored)
 - Scheduling information (priority)
 - Memory management information
 - Accounting information
 - User information
 - Inter-Process Communication (IPC)
 - Other information
- The ID points to the entry in the process table where the pointer to the PCB is stored

Linux Example

```
1164 struct task_struct {  
1165     volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */  
1166     void *stack;  
1167     atomic_t usage;  
1168     unsigned int flags; /* per process flags, defined below */  
1169     unsigned int ptrace;  
1170  
1171 #ifdef CONFIG_SMP  
1172     struct llist_node wake_entry;  
1173     int on_cpu;  
1174     struct task_struct *last_wakee;  
1175     unsigned long wakee_flips;  
.....  
1234 /* task state */  
1235     int exit_state;  
1236     int exit_code, exit_signal;  
1237     int pdeath_signal; /* The signal sent when the parent dies */  
1238     unsigned int jobctl; /* JOBCTL_*, siglock protected */  
1239  
1240     /* Used for emulating ABI behavior of previous Linux versions */  
1241     unsigned int personality;  
1242  
1243     unsigned in_execve:1; /* Tell the LSMs that the process is doing an  
1244             * execve */  
1245     unsigned in_iowait:1;  
1246  
1247     /* task may not gain privileges */  
1248     unsigned no_new_privs:1;  
1249  
1250     /* Revert to default priority/policy when forking */  
1251     unsigned sched_reset_on_fork:1;  
1252     unsigned sched_contributes_to_load:1;  
1253  
1254     pid_t pid;  
1255     pid_t tgid;  
.....  
1577 #ifdef CONFIG_UPROBES  
1578     struct uprobe_task *utask;  
1579 #endif  
1580 #if defined(CONFIG_BCACHE) || defined(CONFIG_BCACHE_MODULE)  
1581     unsigned int sequential_io;  
1582     unsigned int sequential_io_avg;  
1583 #endif  
1584 };
```

RTX Example

```
typedef struct OS_TCB {  
    /* General part: identical for all implementations. */  
    U8 cb_type;      /* Control Block Type */  
    U8 state;        /* Task state */  
    U8 prio;         /* Execution priority */  
    U8 task_id;      /* Task ID value for optimized TCB access */  
  
    struct OS_TCB *p_lnk;    /* Link pointer for ready/sem. wait list */  
    struct OS_TCB *p_rlnk;   /* Link pointer for sem./mbx lst backwards */  
    struct OS_TCB *p_dlnk;   /* Link pointer for delay list */  
    struct OS_TCB *p_blnk;   /* Link pointer for delay list backwards */  
    U16 delta_time;       /* Time until time out */  
    U16 interval_time;    /* Time interval for periodic waits */  
    U16 events;          /* Event flags */  
    U16 waits;           /* Wait flags */  
    void **msg;           /* Direct message passing when task waits */  
  
    struct OS_MUCB *p_mlnk; /* Link pointer for mutex owner list */  
    U8 prio_base;         /* Base priority */  
    U8 ret_val;           /* Return value upon completion of a wait */  
  
    /* Hardware dependant part: specific for CM processor */  
    U8 ret_upd;           /* Updated return value */  
};  
U16 priv_stack;      /* Private stack size, 0= system assigned */  
U32 tsk_stack;        /* Current task Stack pointer (R13) */  
U32 *stack;           /* Pointer to Task Stack memory block */  
  
/* Task entry point used for uVision debugger */  
FUNCP ptask;          /* Task entry address */  
} *P_TCB;
```

- We will come back to this later
- Check with the source code yourself - not really as intimidating as you might expect!

Process Creation

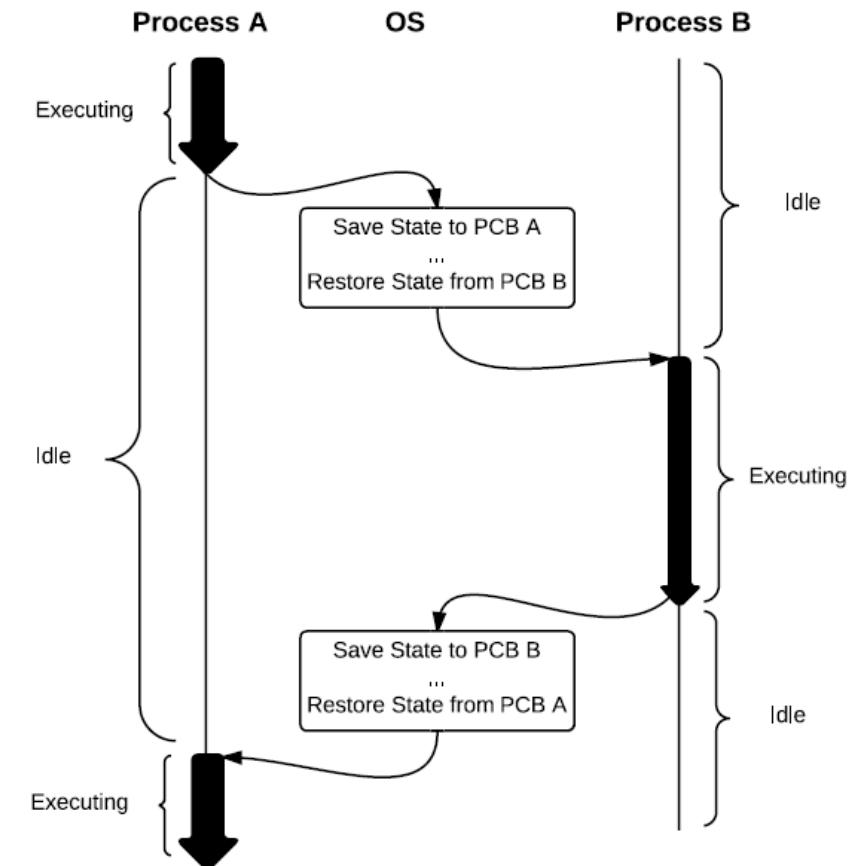
- By initialization, by request of a process, or by request of a user
- Unique ID for the process
 - `init_task_pid()` in Linux
- Allocate memory for the PCB and other control structures (kernel) and user memory
- Initialize the PCB and memory management
- Link the PCB in the queue (see later)

Process Termination

- Stopped by the OS/user (why?) or terminate itself
- Handle the output of the process
- Release the resources and reclaim the memory
- Unlink the PCB
- In embedded software, some processes may never terminate. Terminating implies a fault.

Context Switching

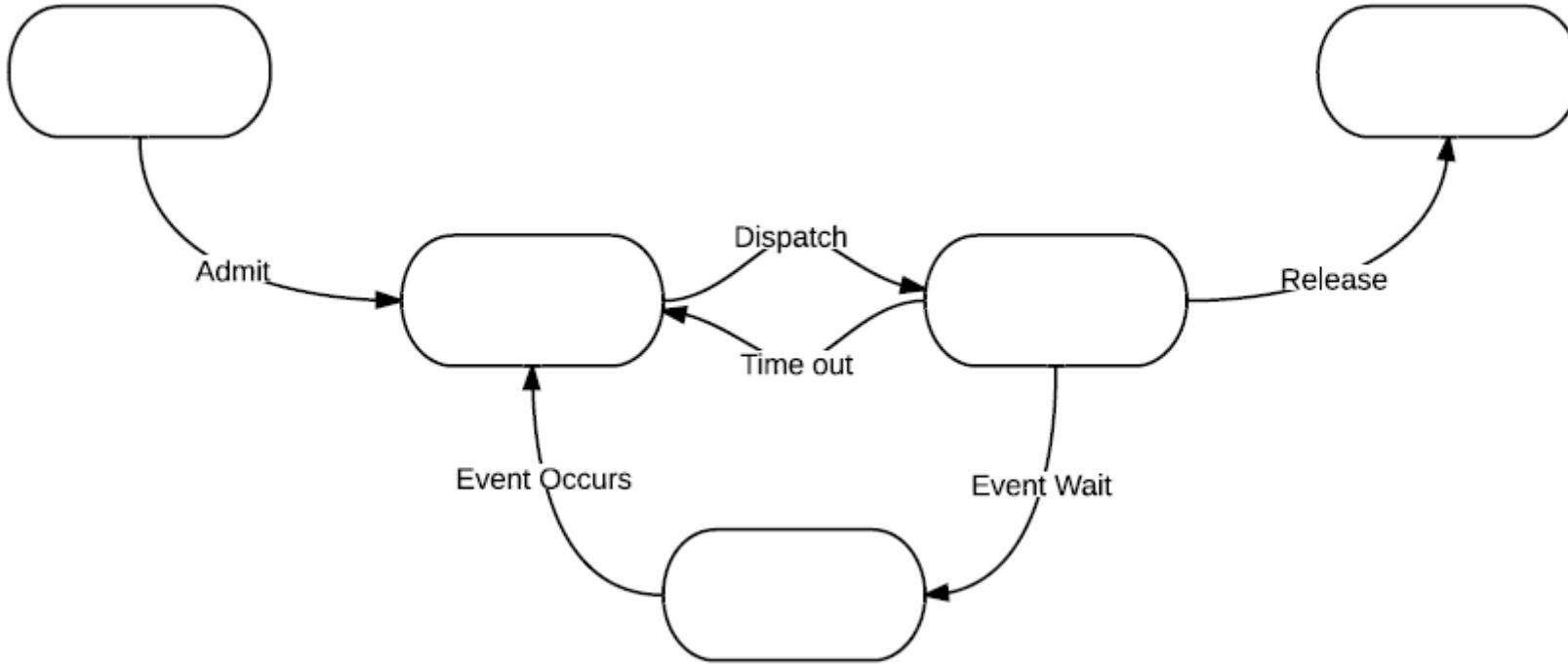
- The PCB makes context switching a bit easier
 - Scheduler will start or stop a process accordingly
 - Stores necessary information in the PCB to stop
 - Hardware registers
 - Program Counter
 - Memory states, stack and heap
 - State
 - Similarly, loads necessary information from the PCB
- Notice that context switching does consume time!
 - Could be up to several thousand CPU cycles
 - Overhead and bottleneck
 - Hardware support is also needed
- Multiprogramming, although only one active process at any given time



Process States

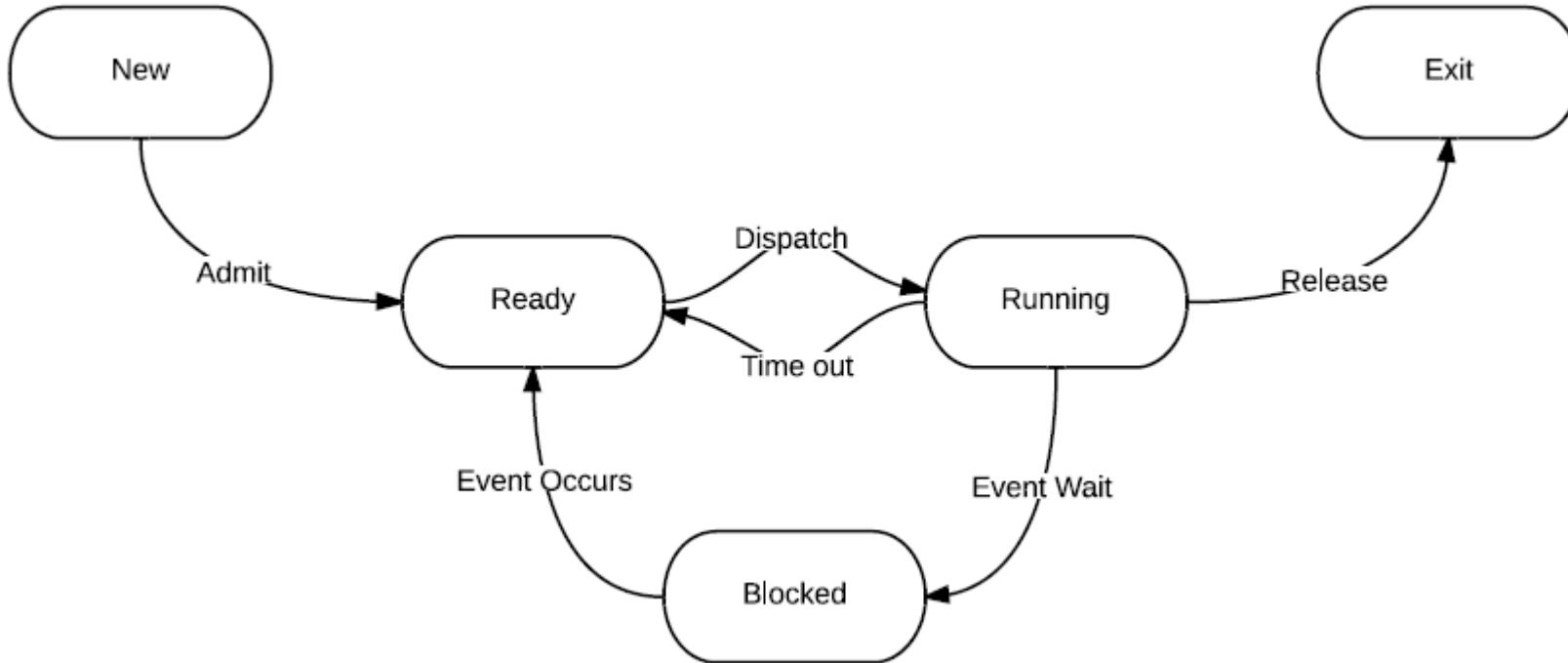
- Different process states during the lifetime cycle
- Many variants but a standard model has five states:
 - New: just been created, not ready for the queue
 - Ready: can be loaded by the OS
 - Running: scheduler has picked this process from the queue and executed it, usually only one
 - Blocked: not in the queue, waiting
 - Exit: finished, needs to terminate
- Linux Example:
 - `TASK_RUNNING`; `TASK_INTERRUPTIBLE`; `TASK_UNINTERRUPTIBLE`; `__TASK_STOPPED`; `__TASK_TRACED`; `EXIT_ZOMBIE`; `EXIT_DEAD`; `TASK_DEAD`; `TASK_WAKEKILL`; `TASK_WAKING`; `TASK_PARKED`; `TASK_STATE_MAX`

State Transition



- State transition as a result of OS scheduling, external interrupts or program requests
- Try to fill in the states in each block

State Transition



- Admit: process is fully loaded into memory and control is established
- Dispatch: scheduler assigns CPU to the process
- Time out: expired or preempted, pushed back to the queue
- Event Wait/Event Occurs: generally requests that cannot be met at the moment, has to wait until something occurs
 - OS not ready for a service
 - Unavailable resource
 - Wait for an input
- Release: release resources and end the process

Process State

- State information is also recorded by the PCB
- Context switch takes place whenever a process leaves/enters the running state
- Processes may make a transition voluntarily or involuntarily, e.g., end the program vs error
- OS typically maintains queue or queue-like (list) structures for processes in the same states (many pointers in the PCB)
 - RTX: rt_list.c
- More complicated models possible:
 - Some processes are stored in the secondary storage in their Ready or Blocked states
 - “Suspended” Ready and Blocked – the seven states model
 - To support **swapping**
 - Scheduler prefer those sit in main the memory

RTX Example

```
/* Values for 'state' */  
#define INACTIVE    0  
#define READY      1  
#define RUNNING    2  
#define WAIT_DLY   3  
#define WAIT_ITV   4  
#define WAIT_OR    5  
#define WAIT_AND   6  
#define WAIT_SEM   7  
#define WAIT_MBX   8  
#define WAIT_MUT   9
```

- No Exit state
- Embedded software don't really enjoy the concept of termination
- More on this in later lectures and labs

Variants of blocked States:
The name indicates the event to invoke the process

Next

- Lab
 - Set up the IDE and get you familiar with C and assembly (if not already)
 - Context switching at processor level
 - Basic RTX task operations
- Scheduling