

Project: Process and Resource Manager

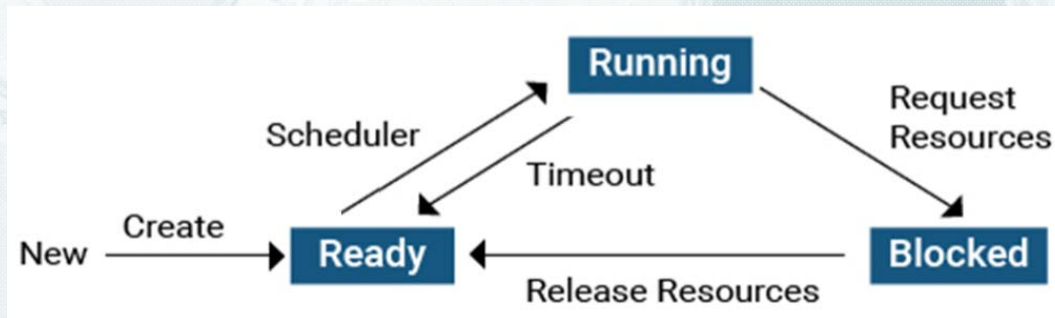
Lubomir Bic
University of California, Irvine

Project Overview

- **Basic Manager** supports
 - data structures to represent and manage processes and resources
 - operations invoked by processes to:
 - create and destroy processes
 - request and release resources
 - timeout function to mimic preemptive scheduling
- Presentation shell allows testing without actual processes and hardware
- **Extended manager**
 - resources can have multiple identical units
 - processes have different priorities and are scheduled accordingly

2.1 Process States

- A process can be in one of three states: **ready, running, blocked**
- Possible state transitions:



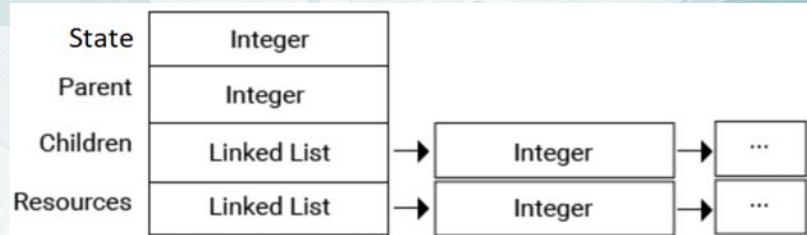
2.2 Representation of Processes

- Each process is represented by a **process control block (PCB)**
- PCBs are organized as a fixed-size array, $PCB[n]$
- Each process is uniquely identified by the PCB index

PCB[n]						
	0	1	...	i	...	n - 1
State						
Parent						
Children						
Resources						

PCB of process i

PCB[i] in basic manager version



- **State:** *running* state can be implicit: head of ready list
 - *ready* and *blocked* are implemented explicitly: integer or binary (1 and 0)
- **Parent:** index of process that created process i
- **Children:** linked list of processes that process i has created
 - each element contains index of child process
- **Resources:** linked list of resources that process i is currently holding
 - each element contains the index of a resource

Lists of processes

- Manager maintains all PCBs on one of several lists:
 - **blocked** processes: kept on **waiting lists** associated with resources
 - **ready** processes: kept on a **Ready List (RL)**
- Basic manager version:
 - all processes have the **same priority**
 - RL is organized as a **single linked list** of PCB indices
- At system **initialization**:
 - process 0 is created automatically and becomes the first running process
 - all other processes are created and destroyed at run time

Process creation

Currently running process, i, can create a new child process, j, using the function:

```
create()
    allocate new PCB[j]
    state = ready
    insert j into list of children of i
    parent = i
    children = NULL
    resources = NULL
    insert j into RL
    display: "process j created"
```

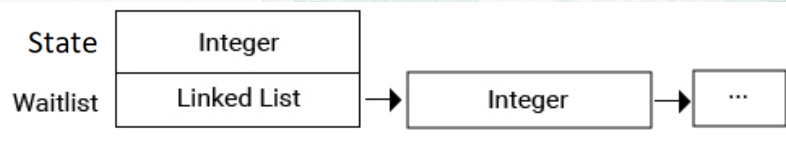
Process destruction

- Currently running process, i, can destroy a child process, j
- The function also recursively destroys all of j's descendants
 - reason for destroying entire subtree: avoid orphan processes

```
destroy(j)
    for all k in children of j destroy(k)
    remove j from list of children of i
    remove j from RL or waiting list
    release all resources of j
    free PCB of j
    display: "n processes destroyed"
```


2.4 Representation of Resources

- System supports a fixed set of resources created at system initialization
- Any process may request, acquire, and later release a resource
- When a resource is unavailable, the requesting process becomes blocked
- Each resource is represented by a **resource control block (RCB)**
- RCBs are organized as a fixed-size array, $RCB[m]$, analogous to PCBs
- Each resource $RCB[r]$ has the form:



- State: free or allocated (1 and 0)
- Waitlist: linked list of processes blocked on the resource

Requesting a resource

Currently running process, i , may request any of the resources, r , at any time:

request(r)

if state of r is free

state of r = allocated

insert r into list of resources of process i

display: "resource r allocated"

else

state of i = blocked

move i from RL to waitlist of r

display: "process i blocked"

scheduler()

Releasing a resource

Currently running process, i , may release any of the resources, r , it is holding:

release(r)

 remove r from resources list of process i

 if waitlist of r is empty

 state of r = free

 else

 move process j from the head of waitlist of r to RL

 state of j = ready

 insert r into resources list of process j

 display: "resource r released"

2.6 Time-Sharing

- Basic manager version: all processes have the **same priority**
 - RL is a single linked list of PCBs accessed in FIFO order
 - process at the head of the RL is the currently running process
- System mimics **preemptive scheduling** by a function timeout()
 - timeout()
 - move process i from head of RL to end of RL
 - scheduler()
- New process, j , now at the head of the RL becomes the running process
- Repeatedly invoking timeout mimics time-sharing

Scheduler

- Scheduler performs **context switch** from currently running process *i* to a new process *j*
 - Scheduler is called whenever:
 - process *i* blocks on a resource and is removed from RL
 - timeout function moves the process to the end of the RL
 - In a real system, context switch:
 - saves CPU state of running process *i*
 - loads CPU state of a new process *j*
 - In this project we do not have physical CPU to save and restore registers
 - scheduler only displays which process is currently running
 - user terminal begins to play the role of currently running process
- scheduler()
- find process *i* currently at the head of RL
- display: "process *i* running"

2.7 System Initialization

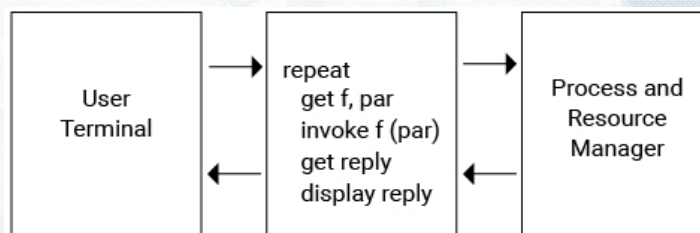
- When system starts, create and initialize the following:
 - PCB[n], all initialized to free except PCB[0]
 - PCB[0] initialized to be running, with no parent, no children, no resources
 - RL contains process 0
 - RCB[m], all initialized to free
- In addition to the functions create, destroy, request, release, timeout:
 - implement *init()* function
 - restores system to original initial state
 - *init* allows continuous testing of the system without having to repeatedly terminate and restart the program

Error handling

- Functions must implement checks to detect **illegal/unexpected operations**
- Examples:
 - Creating more than n processes
 - Destroying a process that is not a child of the current process
 - Requesting a nonexistent resource
 - Requesting a resource the process is already holding
 - Releasing a resource the process is not holding
 - Process 0 should be prevented from requesting any resource to avoid deadlock where no process is on the RL
- In each case, the corresponding function should display “error” and exit

3. The Presentation Shell

- We do not have access to CPU, hardware interrupts, or executable process code
- **Presentation shell:** allows testing and demonstration of manager
 - repeatedly accepts commands from user terminal
 - invokes corresponding manager function
 - displays feedback messages on the screen



- user terminal represents the currently running process
- user terminal also represents the hardware: timeout function is an interrupt

Shell syntax

Shell command	Function
cr	create()
de <i>	destroy(i)
rq <r>	request(r)
rl <r>	release(r)
to	timeout()
in	init()

Example of shell session

```
...
* process 5 running
> cr
* process 6 created
> rq 3
* resource 3 allocated
> to
* process 6 running
> rq 3
* process 6 blocked
* process 5 running
```

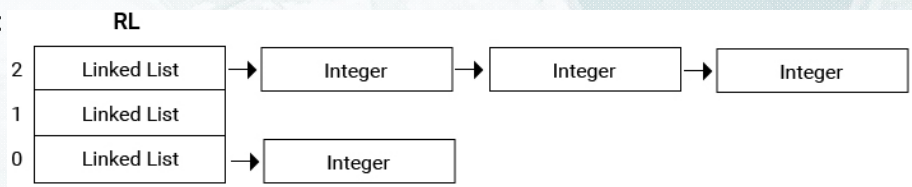
```
> de 6
* 1 process destroyed
> rl 2
* error
> in
* process 0 running
...
```

Exact output format will be specified on course page

4.1 Multilevel Scheduling

- Each process has a fixed **priority**, represented by a positive integer
- RL is extended to segregate processes into FIFO lists according to priorities
- Each list element contains PCB index of the corresponding process
- For this project, RL has 3 priority levels

• **Example:**



- Initially:
 - lowest level, 0, contains 1 process (process 0)
 - priority levels 1 and 2 are empty

Extensions to basic manager

- **PCBs:**
 - new field, **priority**, is included in each PCB: 0, 1, or 2
- **Functions:**
 - **create** function must accept priority value, p, as an argument: *create(p)*
 - p is stored in the priority field of the new PCB
 - process is entered at the corresponding level in RL
 - some functions must call **scheduler** at the end:
 - **create**: context switch if new process has higher priority than current
 - **release**: context switch if *release* unblocks a higher-priority process
 - **delete**: context switch if a deleted process releases a resource on which a higher-level process is blocked

Extensions to basic manager (cont)

scheduler()

find highest priority ready process j

if priority of j > priority of i

display: "process j running"

else

display: "process i running"

- j: head of highest-priority non-empty list
- context switch: if priority of j is greater than priority of current process

• **Shell:**

- cr command must accept priority value: cr p
- additional error checks to deal with priority, e.g., priority > 3 or < 0

4.2. Multiunit Resources

- Each resource is extended to have multiple identical units
- Thus each RCB represents a *class* of resources
- For this project assume 4 resources:
 - Resources 0 and 1 have 1 unit each
 - Resource 2 has 2 units
 - Resource 3 has 3 units

Extensions to basic manager

- **RCBs:**
 - new field, ***inventory***, indicates initial number of units (1, 2, or 3)
 - ***state***: integer counter, to keep track of how many units are still available
 - ***waitlist***: each element contains process index and number of units requested
- **Functions:**
 - ***request***: accept resource index and number of requested units: *request(r, k)*
 - check whether a sufficient number of units is available
 - partial allocation of units should not be supported
 - ***release***: accept resource index and number of released units: *release(r, k)*
 - executes a loop to unblock zero, one, or more processes from waitlist
 - example: releasing 2 units of resource 3 could enable 0, 1, or 2 processes

Extensions to basic manager (cont)

- **Shell:**
 - commands *rq* and *rl* must accept resource index and number of units: "*rq r k*" and "*rl r k*"
 - error checks:
 - number of units requested + number already held \leq initial inventory
 - prevent blocking a process forever
 - number of units released \leq number of units currently held

5. Summary of Specific Tasks

- Design and implement the **extended** version of the manager:
 - PCB, RCB, and RL data structures
 - functions *create()*, *destroy()*, *request()*, *release()*, *timeout()*, *scheduler()*, *init()*
- Design and implement the shell
- Instantiate manager to include the following at start-up:
 - A process descriptor array PCB[16]
 - A single running process, PCB[0], at the lowest-priority level 0
 - A resource descriptor array RCB[4] with multiunit resources
 - A 3-level RL containing process 0 at priority level 0
- Test the manager using a variety of command sequences to explore all aspects

Example 1

Command	Running
	0
cr 1	1
cr 1	1
rq 0 1	1
to	2
rq 1 1	2
rq 0 1	1
rq 1 1	0
de 1	2
to	2