**1 System Overview**

My operating system is designed and implemented in a highly modular fashion, as is shown in Figure 1. While the programming language is C, structures and function pointers are heavily used to achieve the goal of modularity and object-oriented programming. Here, process manager is a good example demonstrating this idea.

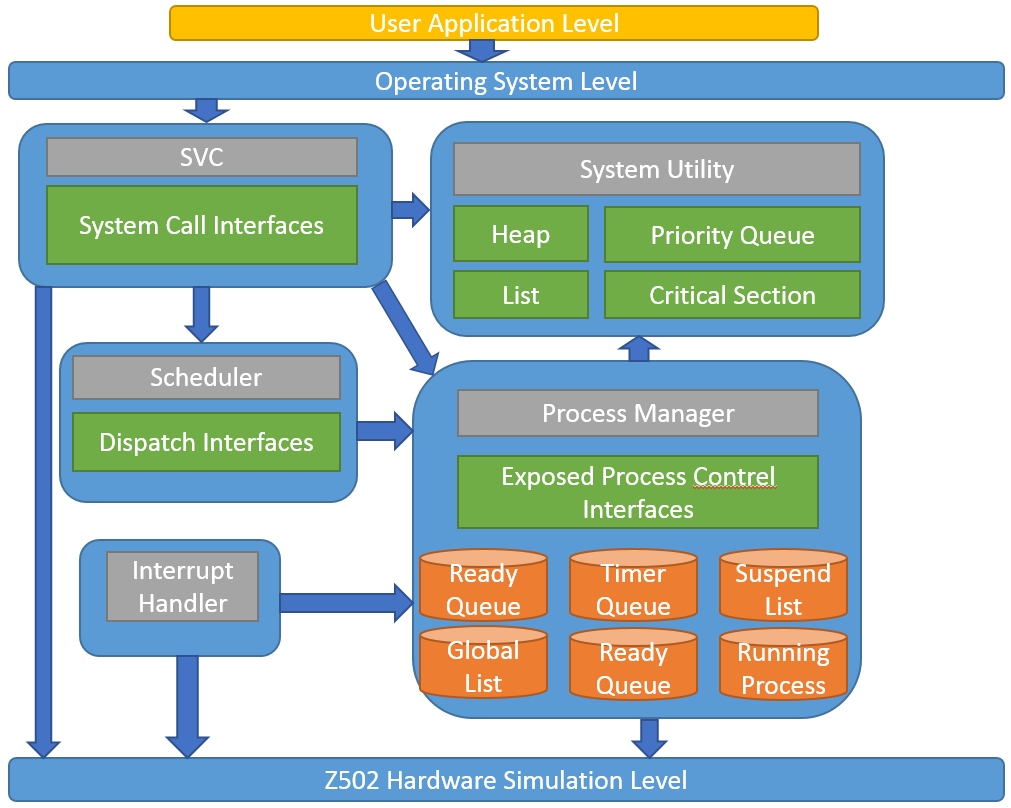


Figure 1. System Overview.

When the operating system starts, it initializes process manager and scheduler, which are singleton objects stored in global variables. Process Manager is in charge of all the process control operations such as create, terminate and suspend a specific process. Other parts of the system must use interfaces exposed by the process manager to do process management related work. In this way, process data management is hidden by the process manager. So other modules do not need to worry about messing up the underlying process queues and lists. In Figure 1, process data are illustrated as orange cylinders.

**2 Modules**

**2.1 SVC**

SVC module is implemented in svc.h and svc.c. Interfaces defined here are:

void SVCGetProcessID(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCTerminateProcess(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCCreateProcess(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCStartTimer(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCSuspendProcess(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCResumeProcess(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCChangeProcessPriority(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCSendMessage(SYSTEM\_CALL\_DATA\* SystemCallData);

void SVCReceiveMessage(SYSTEM\_CALL\_DATA\* SystemCallData);

These functions are implementations of system calls used by user level applications. They do necessary error checks and return calling results to the user level application.

**2.2 Process Manager**

Process Manager module is implemented in process\_manager.h and process\_manager.c. It is a singleton class object created when operating system starts. Here is a bunch of interfaces of it:

// Process manager is a global singleton object used to manage processes.

typedef struct ProcessManager

{

ProcessManagerGetProcessCount GetProcessCount;

ProcessManagerGetTimerQueueProcessCount GetTimerQueueProcessCount;

ProcessManagerGetTimerQueueProcess GetTimerQueueProcess;

ProcessManagerGetReadyQueueProcessCount GetReadyQueueProcessCount;

ProcessManagerGetReadyQueueProcess GetReadyQueueProcess;

.

.

.

ProcessManagerPrintState PrintState;

ProcessManagerResetReadyQueueKeys ResetReadyQueueKeys;

} ProcessManager;

void ProcessManagerInitialize();

void ProcessManagerTerminate();

extern ProcessManager\* gProcessManager;

Here, all interfaces are implemented by using function pointers. This is the only way I know in which object-oriented programming is done using C language. The advantage of doing this is that underlying data such as queues and lists are hidden in the c file. Thus they are invisible to other parts of the system. So it is unlikely that other code mess up the data by directly access. Below are process queues and lists defined in process\_manager.c.

// Global data managed by process manager.

List\* gGlobalProcessList;

MinPriQueue\* gTimerQueue;

MinPriQueue\* gReadyQueue;

List\* gSuspendedList;

PCB\* gRunningProcess;

In my implementation of process ready queue and timer queue, I use heap as underlying data structure. Linked list is easy to manipulate, but insert cost is O(n), whereas heap-based priority queue has a better performance of O(lgn) on average. Every time we pop a PCB object from the heap, it needs O(lgn) to adjust itself. So the overall performance is cO(lgn), which is still O(lgn).

My PCB structure is defined in pcb.h, as follow:

#define PROCESS\_STATE\_UNKNOWN 0

#define PROCESS\_STATE\_READY 1

#define PROCESS\_STATE\_SLEEPING 2

#define PROCESS\_STATE\_RUNNING 3

#define PROCESS\_STATE\_SUSPENDED 4

#define PROCESS\_STATE\_SUSPENDING 5

#define PROCESS\_STATE\_DEAD 6

#define PROCESS\_TYPE\_SCHEDULER 0

#define PROCESS\_TYPE\_USER 1

typedef void(\*ProcessEntry)(void);

typedef struct \_PCB

{

char\* name;

int type; // 0 : scheduler process 1 : user process

long processID;

ProcessEntry entry;

int priority;

int timerQueueKey;

int readyQueueKey;

int state; // 1 : ready 2 : sleep 3 : running

void\* context;

List\* messages;

} PCB;