Documentation of SOS (Strange Operation System)

By

Giorgi Riccardo, Cattani Giovanni, Rrapaj Engjell, Frascolla Stefano

SOS is an evolution of Kaya O.S., itself
evolution of a long list of O.S. proposed for educational
purposes (HOCA, TINA, ICARUS, etc).

This document will explain the specifications of the phase1.

The Phase1 of the SOS has been implemented by managing the process control blocks(pcb) and the semaphore descriptors (semd) means the following modules :

pcb.c – asl.c

(All the functions in these files are recursive)

pcb.c

Module for the initialization, allocation, deallocation and the management of the queues and the trees of pcb

Pcb Lists

With the initPcbs() function we create the list of unused pcb, concatenating the head "pcbFree_h" of this list with the lists of the single pcb through the function freePcb(pcb_t *p).

The available pcb are found in the static vector "pcbtable" which can contain at most "MAXPROC" processes.

The implementation requires that the list of free pcb contains pcb concatenated to each other as a one-way list through the p_next field.

To pick and initiate a pcb from the free list it's necessary to use the function allocPcb() which takes the first element of the list and initiates all of its fields to zero/null.

Pcb Queues

In order to insert a process in a queue we implement the function insertProcQ()which takes in input a double pointer that points at the head of the queue we need to put the process in , and also a "pcb_t *p" that is the ProcBlk that we will insert.

On the other hand in order to remove a process from a queue we implement the removeProcQ() function ,that removes the first element form the queue whose head is pointed by the pointer it takes in input.

If we wanted to remove a process that is not the first process of the queue we call the outProcQ() function which does almost the same as the removeProcQ() but in this case p can point to any element of the process queue

To return the first element of the queue without removing it we use headProcQ() it takes in input the pointer of the queue of whom we want to pick the first pcb .

If we want to call a function for all the pcb in a list, we have to use forallProcQ(), having as input the head of the list, the function we want to call and any arguments.

Pcb Trees

In addition to possibly participating in a process queue, ProcBlk's are also organized into trees of ProcBlk's, called *process trees*. The p_parent, p_child, and p_sib pointers are used for this purpose.
The process trees should be implemented as follows. A parent ProcBlk contains a pointer (p_child) to a NULL-terminated single linearly linked list of its child ProcBlk's. Each child process has a pointer to its parent ProcBlk (p_parent) and possibly the next child ProcBlk of its parent (p_sib).

The functions we use to support process trees are the following

*insertChild(pcb_t *parent, pcb_t *p)*

This function inserts p as one of the childs of the ProcBlk pointed by parent, it first checks if the father has no child, if so it inserts it as the first child, if not it inserts it as the sibling of the next pcb that has a p_sib pointing at NULL

pcb_t removeChild(pcb_t *p)*

This function is needed in order to make remove the first child of the tree pointed by p,it makes the p_first_child to point at p_sib and sets the p_parent of the first child to NULL

*pcb_t * outChild(pcb_t* p)*

In case we needed to remove a child of a tree that it's not its first child then we call the outChild() function, it removes the pcb pointed by p from the list of child of its father, it has another auxiliary function scanSibling() that scans the rest of the tree in order to find the Pcb pointed by p.

asl.c

ASL

In SOS in order to access the shared resources we use the semaphores. Every semaphore is associated to a descriptor (SEMD). In s_key we have the address of the integer variable that contains the value of the semaphore. The address of s_key also is used as identifier of the semaphore.

The semd_table is the array of SEMD with maximum dimension of MAXPROC. The semdFree_h is the head of the free or unused SEMD. Instead semd_h points at the head of the active semaphore list (ASL).

First with the function initASL() we initialize the list of semdFree in order to contain all of the elements in the semd_Table.

The function getSemd(int *key) finds the right semaphore in ASL, having the ID of the semaphore.

The other functions are used to make operations on the list of the pcbs blocked by a semaphore (this list is pointed by s_procQ):

insertBlocked(int *key, pcb_t* p) is used to insert p in the "block list" of the semaphore having ID equal to key (searched in ASL), setting p_semkey with the right value.
If there is no semaphore with that ID, it takes a semd_t from the free list and sets the s_key with key.
If the free list is empty, it returns TRUE; in other cases it returns FALSE.

removeBlocked(int *key) removes the first pcb in the "block list" of the semaphore with ID equal to key, setting p_semkey=NULL.
If it isn't in ASL, returns NULL; else it returns the pcb removed.
Also, if the block list becomes empty, it moves the semaphore from ASL to free list.

headBlocked(int *key) returns the same of removeBlocked, without removing.

outBlocked(pcb_t *p) removes p from the block list of the semaphore where it’s blocked (using p_semkey), then setting p_semkey=NULL.
If the block list becomes empty, it moves the semaphore from ASL to free list.

outChildBlocked(pcb_t *p) does the same things of outBlocked, but also for all the descendants.

forallBlocked(int *key, void fun(struct pcb_t *pcb, void *), void *arg) is a Higher Order function.
It calls fun for all the pcbs blocked by the semaphore with ID equal to key.
To do this, it uses the corrispondent forallprocQ of pcbs’s functions.