Cristiam David Martin Jackson **09.01.2019**

Laboratory Nr. 5 (Report)

"Zarządzanie pamięcią"
"Memory management"

Purpose

The present laboratory consists of the implementation in minix of the algorithm "worst fist" for the management of the memory, allowing to alternate with the standard algorithm (first fit) through the respective system calls, also, implement a function that allows map the memory and consider the available holes and the functionality of the algorithms.

Overview

In order to implement the algorithm worst fit, it is necessary to know its characteristics and the program where it could be implemented. All this process is managed by the server MM, particularly with the function alloc.c, whose function is to allocate and liberate blocks of memory. Its data structure consist of a linked list, whose attributes are, pointer no the block, size and address of the current block.

The worst fit algorithm, consists of the allocation of the biggest block of memory available, with this on mind, the proposed solution traverses the list and compares all the holes looking for the biggest before assigning it to the required process.

Solution

Do_worst_fit, works as selector in order to change the standard algorithm to the one implemented in the "alloc_mem" section.

```
/*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*
 *-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-  LABORATORY 5   =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*
 *-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/

PRIVATE int algorithm = 0; /* 0 (default) for firstFit 1, 1 worstFit */
/*===================================================================*
 *                              worst_fit                            *
 *===================================================================*/
/*Updates global variable algorithm*/
PUBLIC int do_worst_fit(void){
  int sel= mm_in.m1_i1;
  printf("Selected algorithm: %d", sel);
  if (sel == 0 || sel == 1){
    algorithm = sel;
  }
return 0;
}
```

Hole map receives and sends a buffer through system messages, in which the memory will be mapped, in every position is assigned the size of the hole and the address where it begins, the last number added is cero, indicating the end of the buffer.

```

/*===================================================================*
```

```
 *                                hole_map                                *
*========================================================================*/

PUBLIC int do_hole_map(){

phys_clicks buffer[1024];
struct hole * hp;
int i = 0;
int j;
int nbytes = mm_in.m1_i1;/*incoming data size*/
char * buff = mm_in.m1_p1;/*incoming buffer*/
int usr = mm_in.m_source;/*PID*/

if(nbytes%2==0)
        nbytes-=2; /* reserve space for 0 (end of buffer) */
j = nbytes/2;

hp = hole_head;
if(hp == NIL_HOLE)
        return -1;
do{
        buffer[i++]=hp->h_len; /* length */
/*      printf("length of this hole: %d\n",hp->h_len);*/
/*      printf("clickscdmj: %d\n",clickscdmj);*/
        buffer[i++]=hp->h_base; /* address */
        hp=hp->h_next; /* next hole */
        j--;
}
while(hp != NIL_HOLE && hp->h_base < swap_base && j > 0);
buffer[i]=0; /* last position of the buffer, write 0 to finish */
sys_copy(MM_PROC_NR,D,(phys_bytes) buffer, usr, D, (phys_bytes) buff, nbytes);
return i/2;/*Number of transferred pairs*/
}
```

Worst fit algorithm

It traverses the full list of free blocks, in order to pick the biggest one, once it is found, the algorithm compares if it is suitable for the amount of memory required.

```
else {/*ALGORITH for WORSTFIT*/

        register struct hole *big;
        hp = hole_head;
/*      printf("clicks: %d\n",clicks);*/
        /*initialize biggest if first hole in the list is bigger enough*/
        /*
        if (hp != NIL_HOLE && hp->h_len>=clicks)
                biggest = hp->h_base;
        else
                return (NO_MEM);*/
        /*There is no hole bigger enough*/
```

```
        while (hp != NIL_HOLE && hp->h_base < swap_base){/*traverse the list in order to
find the biggest hole*/
                if (hp->h_len>biggest){
                        biggest = hp->h_len;
                        old_base = hp->h_base;/*starting point of the current biggest one*/
                        big=hp;
                }
                prev_ptr = hp;
                hp = hp->h_next;
        }
        if (big->h_len >= clicks) {
                        /* We found the biggest hole and is suitable.  Use it. */
                        old_base = big->h_base;       /* remember where it started */
                        big->h_base += clicks;          /* bite a piece off */
                        big->h_len -= clicks;   /* ditto */
                        /* Delete the hole if used up completely. */
                        if (big->h_len == 0)
                                del_slot(prev_ptr, big);
                        /* Return the start address of the acquired block. */
                        return(old_base);
        }
        else
                return (NO_MEM);/*If the hole was not biggest enough*/
  }/*else (worstFit)*/
```

The implemented testing functions where provided in the laboratory's page. They consist of three files: t.c, x.c and w.c, for mapping the memory, run background process and select the algorithm.

Results

FIRST-FIT ALGORITHM

The next table shows the holes present in the memory during the test, during 10 iterations the file x.c was run in the background, the next 10 iterations it stop running in the background. It can be appreciated and assumed, according to the patterns in the first 10 rows, that the x.c file required 9 clicks of memory, after the 10th iteration the memory started reallocating hole by hole.

This table also allows to appreciate how does first-fit algorithm work, where the first hole with the capacity to hold 9 clicks was assigned.

```
# ./script.sh
x: Stack+malloc area changed from 131072 to 8000 bytes.
-[ std ]----------------------------------------
Selected algorithm: 0[5]        (68) (6) (21) (90) (128563)
[5]     (68) (6) (12) (90) (128563)
[5]     (68) (6) (3) (90) (128563)
[5]     (68) (6) (3) (81) (128563)
[5]     (68) (6) (3) (72) (128563)
[5]     (68) (6) (3) (63) (128563)
[5]     (68) (6) (3) (54) (128563)
[5]     (68) (6) (3) (45) (128563)
[5]     (68) (6) (3) (36) (128563)
[5]     (68) (6) (3) (27) (128563)
[5]     (68) (15) (3) (27) (128563)
[6]     (68) (15) (9) (3) (27) (128563)
[5]     (68) (15) (21) (27) (128563)
[6]     (68) (15) (21) (9) (27) (128563)
[6]     (68) (15) (21) (18) (27) (128563)
[6]     (68) (15) (21) (27) (27) (128563)
[6]     (68) (15) (21) (36) (27) (128563)
[6]     (68) (15) (21) (45) (27) (128563)
[6]     (68) (15) (21) (54) (27) (128563)
[5]     (68) (17) (21) (90) (128563)
```

	68	6	21	90	128563			0	0	9	0	0
[5]	68	6	12	90	128563			0	0	9	0	0
[5]	68	6	3	90	128563			0	0	9	0	0
[5]	68	6	3	81	128563			0	0	0	9	0
[5]	68	6	3	72	128563			0	0	0	9	0
[5]	68	6	3	63	128563			0	0	0	9	0
[5]	68	6	3	54	128563			0	0	0	9	0
[5]	68	6	3	45	128563			0	0	0	9	0
[5]	68	6	3	36	128563			0	0	0	9	0
[5]	68	6	3	27	128563			0	0	0	9	0
[5]	68	15	3	27	128563			0	-9	0	0	0
[6]	68	15	9	3	27	128563		0	0	-6	24	128536
[5]	68	15	21	27	128563			0	0	-12	-24	-128536
[6]	68	15	21	9	27	128563		0	0	0	18	128536
[6]	68	15	21	18	27	128563		0	0	0	-9	0
[6]	68	15	21	27	27	128563		0	0	0	-9	0
[6]	68	15	21	36	27	128563		0	0	0	-9	0
[6]	68	15	21	45	27	128563		0	0	0	-9	0
[6]	68	15	21	54	27	128563		0	0	0	-9	0
[5]	68	17	21	90	128563			0	-2	0	-36	-128536

Some extra test was done, printing the size of clicks every time the allocation function (in alloc.c) was executed, the preview is shown ahead.

Beginning of the algorithm first fit: the size of 9 clicks is appreciated.

```
clicks first fit: 111
clicks first fit: 144
clicks first fit: 62
clicks first fit: 26
x: Stack+malloc area changed from 131072 to 8000 bytes.
-[ std ]---------------------------------------
clicks first fit: 62
clicks first fit: 130
Selected algorithm: 0clicks first fit: 62
clicks first fit: 11
clicks first fit: 62
clicks first fit: 135
[5]     (68) (6) (21) (90) (128563)
clicks first fit: 62
clicks first fit: 19
clicks first fit: 62
clicks first fit: 9
clicks first fit: 62
clicks first fit: 135
[5]     (68) (6) (12) (90) (128563)
clicks first fit: 62
clicks first fit: 19
```

Ending of the first-fit algorithm:

```
clicks first fit: 135
[6]     (68) (15) (21) (45) (27) (128563)
clicks first fit: 62
clicks first fit: 19
clicks first fit: 62
clicks first fit: 135
[6]     (68) (15) (21) (54) (27) (128563)
clicks first fit: 62
clicks first fit: 19
clicks first fit: 62
clicks first fit: 135
[5]     (68) (17) (21) (90) (128563)
clicks first fit: 62
clicks first fit: 19
-[ worst ]-------------------------------------
clicks first fit: 62
clicks first fit: 130
Selected algorithm: 1clicks 62
 clicks 11
 clicks 62
 clicks 135
 [7]    (68) (17) (21) (90) (62) (62) (128428)
clicks 62
 clicks 19
```

From the table can be appreciated a difference of 11 clicks in the sizes of the holes at the beginning and at the end of the first fit algorithm. It is due to the allocation of memory that is done as soon as the first fit algorithm is selected and before the map for the first iteration is obtained.

```
x: Stack+malloc area changed from 131072 to 8000 bytes.
-[ std ]---------------------------------------
clicks first fit: 62
clicks first fit: 130
Selected algorithm: 0clicks first fit: 62
clicks first fit: 11
clicks first fit: 62
clicks first fit: 135
[5]     (68) (6) (21) (90) (128563)
```

WORST-FIT ALGORITHM

Up next, the results of the worst-fit algorithm are shown, first of all, the working of the algorithm is appreciated, in every one of the ten iterations, the hole with the biggest size was chosen.

```
-[ worst ]-------------------------------------
Selected algorithm: 1[7]        (68) (17) (21) (90) (62) (62) (128428)
[8]     (68) (17) (21) (90) (62) (62) (62) (128357)
[9]     (68) (17) (21) (90) (62) (62) (62) (62) (128286)
[10]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (128215)
[11]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (128144)
[12]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (62) (128073)
[13]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (62) (62) (128002)
[14]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (62) (62) (62) (127931
)
[15]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (62) (62) (62) (62) (1
27860)
[16]    (68) (17) (21) (90) (62) (62) (62) (62) (62) (62) (62) (62) (62) (62) (6
2) (127789)
[16]    (68) (17) (21) (90) (62) (71) (62) (62) (62) (62) (62) (62) (62) (62) (6
2) (127789)
[15]    (68) (17) (21) (90) (62) (142) (62) (62) (62) (62) (62) (62) (62) (62) (
127789)
[14]    (68) (17) (21) (90) (62) (213) (62) (62) (62) (62) (62) (62) (62) (12778
9)
[13]    (68) (17) (21) (90) (62) (284) (62) (62) (62) (62) (62) (62) (127789)
[12]    (68) (17) (21) (90) (62) (355) (62) (62) (62) (62) (62) (127789)
[11]    (68) (17) (21) (90) (62) (426) (62) (62) (62) (62) (127789)
```

From the 10th iteration on, holes start merging from different address of memory, due to the merge algorithm (which was no modified).

[7]	68	17	21	90	62	62	128428										128428	
[8]	68	17	21	90	62	62	62	128357									128357	71
[9]	68	17	21	90	62	62	62	62	128286								128286	71
[10]	68	17	21	90	62	62	62	62	62	128215							128215	71
[11]	68	17	21	90	62	62	62	62	62	62	128144						128144	71
[12]	68	17	21	90	62	62	62	62	62	62	62	128073					128073	71
[13]	68	17	21	90	62	62	62	62	62	62	62	62	128002				128002	71
[14]	68	17	21	90	62	62	62	62	62	62	62	62	62	127931			127931	71
[15]	68	17	21	90	62	62	62	62	62	62	62	62	62	62	127860		127860	71
[16]	68	17	21	90	62	62	62	62	62	62	62	62	62	62	62	127789	127789	71
[16]	68	17	21	90	62	71	62	62	62	62	62	62	62	62	62	127789	127789	0
[15]	68	17	21	90	62	142	62	62	62	62	62	62	62	62	127789		127789	0
[14]	68	17	21	90	62	213	62	62	62	62	62	62	62	127789			127789	0
[13]	68	17	21	90	62	284	62	62	62	62	62	62	127789				127789	0
[12]	68	17	21	90	62	355	62	62	62	62	62	127789					127789	0
[11]	68	17	21	90	62	426	62	62	62	62	127789						127789	0
[10]	68	17	21	90	62	497	62	62	62	127789							127789	0
[9]	68	17	21	90	62	568	62	62	127789								127789	0
[8]	68	17	21	90	62	639	62	127789									127789	0
[6]	68	17	21	90	62	128501												

Since the background process was the same as the one executed for the first fit algorithm, it is expected to obtain the same required amount of memory, the difference lies in the way it is allocated, it can be possible that two allocated blocks of memory are found one next to other, in that case, it can explain why the difference shows 71 clicks. It can be seen clearly in the next image.

```
-[ worst ]-------------------------------------
clicks first fit: 62
clicks first fit: 130
Selected algorithm: 1clicks 62
 clicks 11
 clicks 62
 clicks 135
 [7]    (68) (17) (21) (90) (62) (62) (128428)
clicks 62
 clicks 19
 clicks 62
 clicks 9
 clicks 62
 clicks 135
 [8]    (68) (17) (21) (90) (62) (62) (62) (128357)
clicks 62
 clicks 19
 clicks 62
 clicks 9
 clicks 62
 clicks 135
 [9]    (68) (17) (21) (90) (62) (62) (62) (62) (128286)
clicks 62
 clicks 19
```

In similar way as to the first fit algorithm, the amount of memory (in clicks) required for the operating system during the test is shown. It is appreciated the requirement of 9 clicks (which I assume is what the process given by x.c needs), as well as different values of clicks that the memory needs to manage.

The difference in the memory size (11 clicks), is due to the same reasons as for the first-algorithm, there is an allocation of 11 clicks of memory right before the first iteration (the first map operation of memory) is done.

```
-[ worst ]-------------------------------------
clicks first fit: 62
clicks first fit: 130
Selected algorithm: 1clicks 62
 clicks 11
 clicks 62
 clicks 135
 [7]    (68) (17) (21) (90) (62) (62) (128428)
```

As clearest evidence, It can be appreciated a pattern in the clicks needed during first-fit test, as well as during worst-fit test, those amounts of required memory are:

During first 10 iterations	**From 11th iteration on**
(clicks)	**(clicks)**
135	135
62	62
19	19
62	62
9	.
.	.
.	.
.	