Fundamentos de los Sistemas Operativos (FSO)

Departamento de Informática de Sistemas y Computadoras (DISCA)
Universitat Politècnica de València

Memory management

Challenge

fSO

- This program defines 3 MM policies (FF, BF and WF) to handle a variable memory scheme.
- Memory class offers a set of operations to handle the memory space
  - **defineMemory**(size, mode, strategy)
    - Size: int => amount of memory
    - Mode: boolean => coalescence/no coalescence
    - Strategy: string => FF|BF|MM
  - **setPolicy**(npolicy) redefines the policy
  - **allocProcess**(pid, size) allocates a process in memory. Returns True if process is allocatef or False if failed
  - **deallocProcess**(pid) removes a process from memory
  - memMap(), memInfo(), memShow()

- FSO-MM can be invoked witht he following options

Usage: python FSO-MM.py [-t ticks] [-p FF|BF|WF] [-u util] [-s seed] [-c] [-v]

default values

p	FF	policy
t	100	number of time ticks
u	60	utilization
c		if flag set => No coalescence
s	None	seed initialization for randomize
v		if flag set => verbose

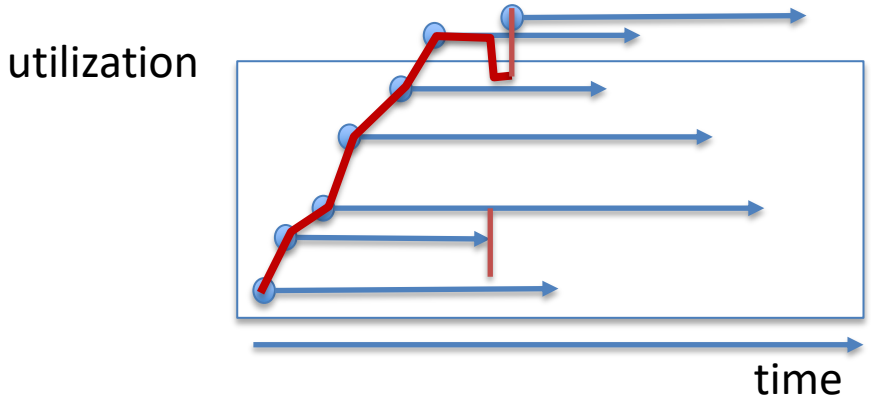
```
$ python FSO-MM.py
Policy: FF Holes:(max, min, sum) = 2860, 170, 4340
Memory:(used, frag) = 57.617, 34.101

$ python FSO-MM.py -p WF -t 1000 -u 60 -s 1234567
Policy: WF Holes:(max, min, sum) = 1030, 500, 4020
Memory:(used, frag) = 60.742, 74.378
```

- FSO-MM can be invoked with the following options

```
$ python FSO-MM.py -p WF -t 50 -u 30 -s 1234567 -c -v
0 Allocated  P001 Size:  580 ending: 19 True 5.6640625
1 Allocated  P002 Size:  390 ending: 19 True 9.47265625
2 Allocated  P003 Size:  580 ending: 35 True 15.13671875
3 Allocated  P004 Size:  130 ending: 31 True 16.40625
4 Allocated  P005 Size:  400 ending: 42 True 20.3125
5 Allocated  P006 Size:  510 ending: 17 True 25.29296875
6 Allocated  P007 Size:  570 ending: 42 True 30.859375
19 Deallocated  P001 25.1953125
19 Allocated  P008 Size:  440 ending: 32 True 29.4921875
20 Deallocated  P002 25.68359375
20 Allocated  P009 Size:  220 ending: 24 True 27.83203125
.........
44 Allocated  P014 Size:  420 ending: 30 True 33.7890625
46 Deallocated  P005 29.8828125
46 Allocated  P015 Size:  350 ending: 43 True 33.30078125
48 Deallocated  P007 27.734375
48 Allocated  P016 Size:  400 ending: 45 True 31.640625
Policy: WF Holes:(max, min, sum) = 3620, 130, 7000 Memory:(used, frag) = 31.641, 48.286
Policy: WF Holes:(max, min, sum) = 3620, 130, 7000 Memory:(used, frag) = 31.641, 48.286
Policy: WF Holes:  [(0, 580), (580, 390), (970, 580), (1550, 130), (1680, 400), (2080, 510), (2590, 570), (3600, 220), (6620,
3620)]
EMP0, 0, 580, 580
EMP1, 580, 390, 970
EMP2, 970, 580, 1550
EMP3, 1550, 130, 1680
EMP4, 1680, 400, 2080
.......
P012, 4610, 300, 4910
P013, 4910, 540, 5450
P014, 5450, 420, 5870
P015, 5870, 350, 6220
P016, 6220, 400, 6620
EMP8, 6620, 3620, 10240
```

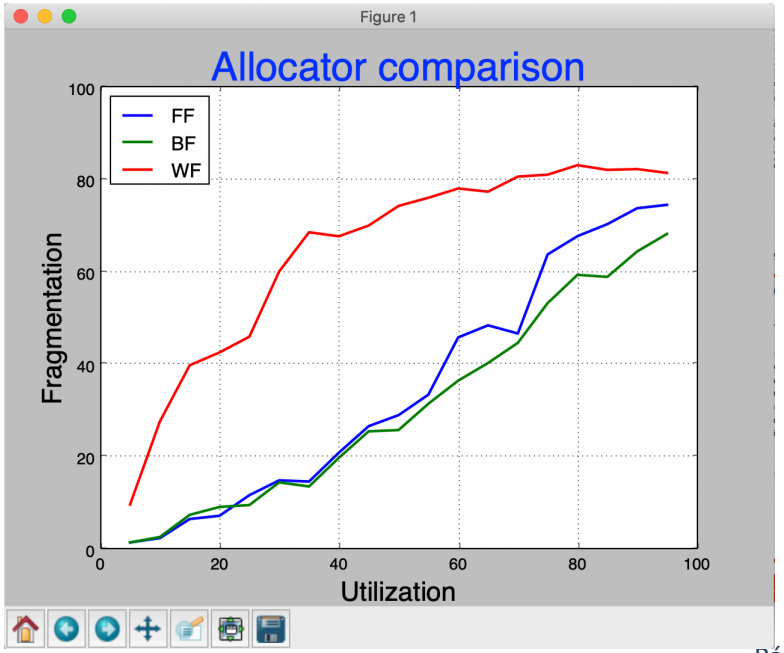
```
Usage: python FSO-MM.py [-t ticks] [-p FF|BF|WF] [-u util] [-s seed] [-c] [-v]
    default values
     p  FF     policy
     t  100    number of time ticks
     u  60     utilization
     c         if flag set => No coalescence
     s  None   seed initialization for randomize
     v         if flag set => verbose
```



- Challenge 1
  - Comparison of 3 policies. A set of random load in the range of utilization.

- $ **python Compare.py –x**
- **Usage: python Compare.py [-n ntests] [-u utilization] [-i increment] [-t nticks]**
- **default values**
- **n 10 number of tests**
- **t 100 number of time ticks**
- **u 10 initial utilization**
- **i 10 utilization increment**

- **Example:**
- **$ python Compare.py -n 20 -u 5 -i 5 -t 1000**



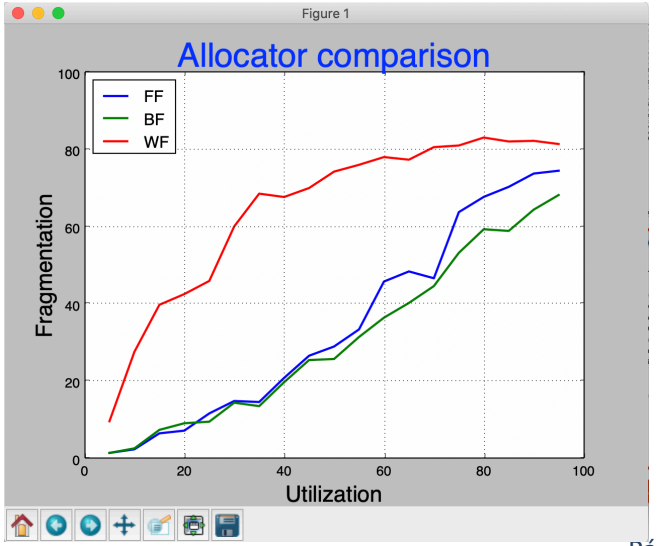
- Challenge 1
  - Comparison of 3 policies. A set of random load in the range of utilization.

A random scenario is a sequence of random processes to be allocated in memory using a memory allocator algorithm. In order to compare the fragmentation produced by each allocator it is desirable that all of them use the same random scenario.

A set of test is defined as several random scenarios to be applied to the allocators.

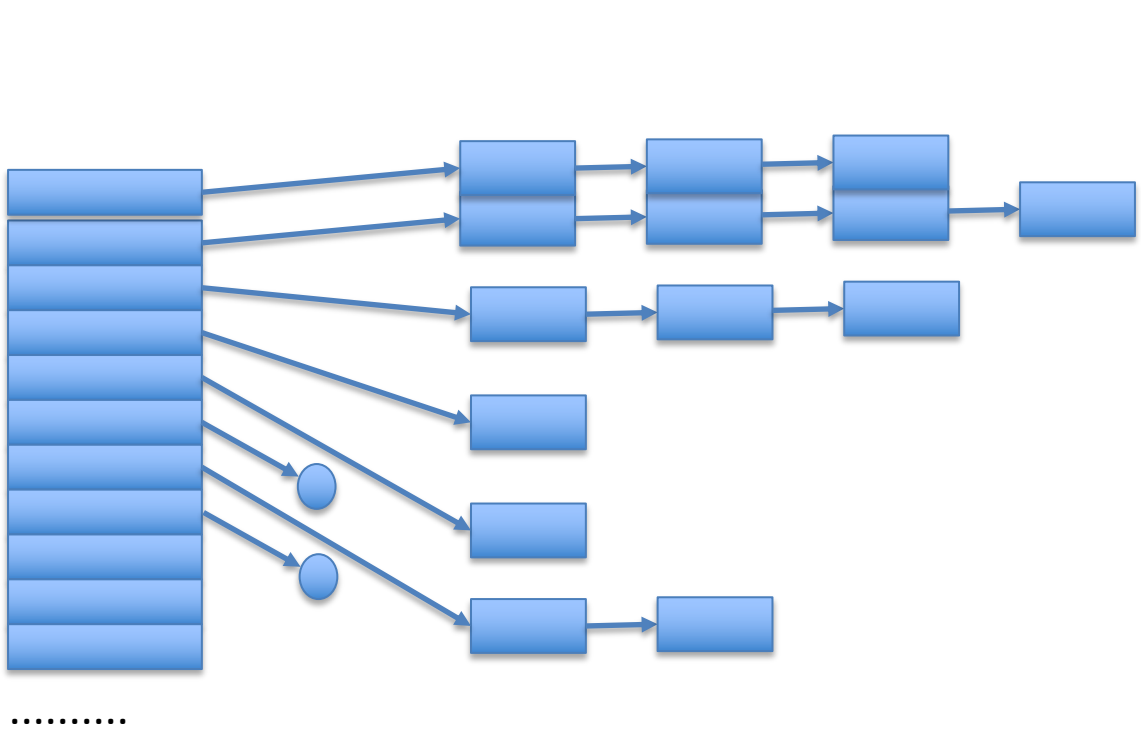
Each set of test has in common the utilization target of memory.

Compare script has to draw the comparison of the fragmentation between all allocators when a set of test specified in the command parameters is executed for a range of utilization beetwen [initial_utilization, final_utilization (<100)] using an utilization increment specified.



- Challenge 2
- Algorithm: Segregated List

0	31	32	2^5
32	63	64	2^6
64	127	128	2^7
128	255	256	2^8
256	511	512	2^9
512	1023	1024	2^10
1024	2047	2048	2^11
2048	4095	4096	2^12
4096	8191	8192	2^13
8192	16383	16384	2^14
16384	32767	32768	2^15
32768	65535	65536	2^16
65536	131071	1E+05	2^17



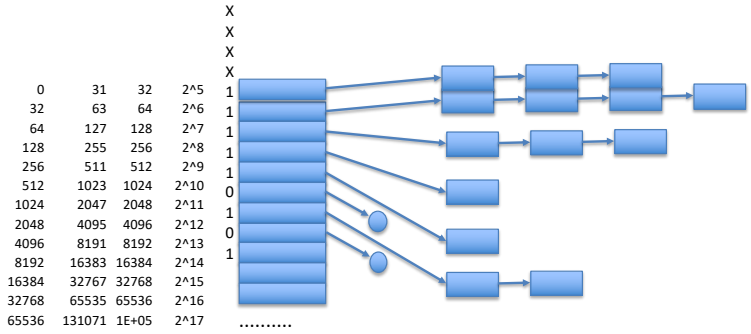
bitmap 110101111xxxxxx

• Challenge 2

Segregated list algorithm uses an array of hole (free) lists where each array contains blocks of the same size or class size (i.e. power of two).

Class 5: all holes of size [>0 and <32]
Class 6: all holes of size [2^6-1, 2^5]
...
Class n: all holes of size [2^n-1, 2^{n-1}]. n < 22

A bitmap word is used to identify when a class is empty or not.



Method:
Allocate a process of size S implies to identify the class that better fits the need and use the first hole in the list. The remaining hole when allocated, has to be added at the beginning of its class.
If coalescence is true, it has to merge with its neighboring holes before adding to its class.
Remove a process, means to free the occupied memory block, coalesce or not according to the selected option, and added to the corresponding class.

All operations related to Class 6 or higher have to be constant which implies that no loop is allowed.

A different algorithm can be applied when size is lower than 32 bits. In that case, you can use BF. The allocation is complete no split of the hole is allowed. It means that if the request is 28 bytes and the available hole higher than 28 is 30, all the hole is allocated to the request.

When a request can not be attended the response of the allocator is false else true.