9-cpu-sched:

1.

Ba FIFO:

```
./scheduler.py -p FIFO -l 200,200,200 -c
turnaround = 400
response = 200
```

Ba SJF:

```
./scheduler.py -p SJF -l 200,200,200 -c
turnaround = 400
response = 200
```

2.

Ba FIFO:

```
./scheduler.py -p FIFO -l 100,200,300 -c
turnaround = 366.6
response = 166.6
```

Ba SJF:

```
./scheduler.py -p SJF -l 100,200,300 -c
turnaround = 333.3
response = 133.3
```

3.

RR:

```
./scheduler.py -p RR -q 1 -l 100,200,300 -c
turnaround = 466.6
response = 2
```

4.

برای او دسته از تسک هایی که طول تسک ها برابرند یا در یک زمان می رسند.

5.

برای موقعی که اندازه کوانتومی که برای rr است برابر اندازه هر تسک باشد.

6.

ریسپانس تایم افزایش می یابد.

```
./scheduler.py -l 1,1,1 -p SJF -c
./scheduler.py -l 2,2,2 -p SJF -c
./scheduler.py -l 3,3,3 -p SJF -c
```

7.

به تبع این اتفاق ریسپانس تایم افزایش می یابد.

معادله بدترین حالت:

$$sum(0\ to\ N)\ *\ ((quantum\ ...)/N)$$

18-Segmentation:

1.

1.1)

```
Virtual Address Trace
  VA  0: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:
492)
  VA  1: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  2: 0x00000035 (decimal:   53) --> SEGMENTATION VIOLATION (SEG0)
  VA  3: 0x00000021 (decimal:   33) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x00000041 (decimal:   65) --> SEGMENTATION VIOLATION (SEG1)
```

1.2)

```
Virtual Address Trace
  VA  0: 0x00000011 (decimal:   17) --> VALID in SEG0: 0x00000011 (decimal:
17)
  VA  1: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:
492)
  VA  2: 0x00000061 (decimal:   97) --> SEGMENTATION VIOLATION (SEG1)
  VA  3: 0x00000020 (decimal:   32) --> SEGMENTATION VIOLATION (SEG0)
  VA  4: 0x0000003f (decimal:   63) --> SEGMENTATION VIOLATION (SEG0)
```

1.3)

```
Virtual Address Trace
  VA  0: 0x0000007a (decimal:  122) --> VALID in SEG1: 0x000001fa (decimal:
506)
  VA  1: 0x00000079 (decimal:  121) --> VALID in SEG1: 0x000001f9 (decimal:
505)
  VA  2: 0x00000007 (decimal:    7) --> VALID in SEG0: 0x00000007 (decimal:
7)
  VA  3: 0x0000000a (decimal:   10) --> VALID in SEG0: 0x0000000a (decimal:
10)
  VA  4: 0x0000006a (decimal:  106) --> SEGMENTATION VIOLATION (SEG1)
```

2.

بالاترین ادرس مجاز توی سگمنت ۰ برابر ۱۹ است و پایین ترین ادرس مجاز توی سگمنت ۱ برابر ۱۰۸ است توی کل فضای ادرس بالا ترین و پایین ترین ادرس به ترتیب برابر ۱۲۷و ۰ است

برای تست هم کافی هست یکبار مرز ها را بررسی کنیم و چند تا هم داخل مرز:

```
./segmentation.py -a 128 -p 512 -b 0 -l 20 -B 512
-L 20 -s 2 -A 127,126,108,107,106,20,19,18,0 -c
```

Result:

```
Virtual Address Trace
  VA  0: 0x0000007f (decimal:  127) --> VALID in SEG1: 0x000001ff (decimal:
511)
  VA  1: 0x0000007e (decimal:  126) --> VALID in SEG1: 0x000001fe (decimal:
510)
  VA  2: 0x0000006c (decimal:  108) --> VALID in SEG1: 0x000001ec (decimal:
492)
  VA  3: 0x0000006b (decimal:  107) --> SEGMENTATION VIOLATION (SEG1)
  VA  4: 0x0000006a (decimal:  106) --> SEGMENTATION VIOLATION (SEG1)
  VA  5: 0x00000014 (decimal:   20) --> SEGMENTATION VIOLATION (SEG0)
  VA  6: 0x00000013 (decimal:   19) --> VALID in SEG0: 0x00000013 (decimal:
19)
  VA  7: 0x00000012 (decimal:   18) --> VALID in SEG0: 0x00000012 (decimal:
18)
  VA  8: 0x00000000 (decimal:    0) --> VALID in SEG0: 0x00000000 (decimal:
0)
```

3.

کافی است مقدار های زیر را دهیم:

--b0=0 --l0=2 --b1=16 --l1=2

4.

./segmentation.py -a 90 -p 100

5.

بهترین کار این است که پارامتر های ال و بی را صفر بگذاریم که هیچ آدرسی در محدوده نباشد.

مثلا:

./segmentation.py --l0=0 --l1=0

19-Freespace

1.

بنظر میرسد که طی زمان لیست مکان های آزاد شده زیاد و قطعه قطعه می شود.

```
ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92
]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84
]
```

```
Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 4 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84
]

Free(ptr[3]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 4 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ]
[ addr:1016 sz:84 ]
```

2.

به نظر می رسد که لیست مکان های ازاد طولانی تر شده است چون توی این استراتژی اول بزرگترین مکان پر میشود پس مکان ها کمتر باز مورد استفاده قرار خواهند گرفت.

```
ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92
]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84
]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1016 (searched 4 elements)
```

```
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1024 sz:76 ]

Free(ptr[3]) returned 0
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:8 ] [ addr:1024 sz:76 ]

ptr[4] = Alloc(2)  returned 1024 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:8 ] [ addr:1026 sz:74 ]

ptr[5] = Alloc(7)  returned 1026 (searched 5 elements)
Free List [ Size 5 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:8 ] [ addr:1033 sz:67 ]
```

3.

چون پالسی به این صورت است که به اولین خانه رسیدیم ان را پر می کنیم پس تایم الوکیشن بهتری میشود (یعنی سریعتر میشود).

```
ptr[0] = Alloc(3)  returned 1000 (searched 1 elements)
Free List [ Size 1 ]:  [ addr:1003 sz:97 ]

Free(ptr[0]) returned 0
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:97 ]

ptr[1] = Alloc(5)  returned 1003 (searched 2 elements)
Free List [ Size 2 ]:  [ addr:1000 sz:3 ] [ addr:1008 sz:92 ]

Free(ptr[1]) returned 0
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:92
]

ptr[2] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84
]

Free(ptr[2]) returned 0
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[3] = Alloc(8)  returned 1008 (searched 3 elements)
Free List [ Size 3 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1016 sz:84
]

Free(ptr[3]) returned 0
```

```
Free List [ Size 4 ]:  [ addr:1000 sz:3 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[4] = Alloc(2)  returned 1000 (searched 1 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1008 sz:8 ]
[ addr:1016 sz:84 ]

ptr[5] = Alloc(7)  returned 1008 (searched 3 elements)
Free List [ Size 4 ]:  [ addr:1002 sz:1 ] [ addr:1003 sz:5 ] [ addr:1015 sz:1 ]
[ addr:1016 sz:84 ]
```

4.

```
./malloc.py -p FIRST -l SIZESORT+ -c
./malloc.py -p BEST -l SIZESORT+ -c
./malloc.py -p WORST -l SIZESORT- -c
```

5.

فرست فیت ظاهرا طول لیست مکان های ازاد را افزایش می دهد.
اما در سناریو های بست و ورست ظاهرا ترتیب در لیست به صورت المنت های کوچک تر است از طرفی این قضیه سرعت الوکیشن را بالاتر می برد. از طرفی هم سورت کردن ادرس ها بهتر است.

6.

توی حالت اول بعضی جاها -۱ بر میگرداند که نشان دهنده این است که موری برای الوکیت تمام شده است توی حالت دوم هم لست مکان های ازاد همیشه طولی برابر ۱ دارد.

7.

مثلا :

```
./malloc.py -n 100 -p WORST -P 73 -c
./malloc.py -n 100 -p FIRST -l SIZESORT- -P 68 -c
```

21-vm-tlbs:

1. اگه از تابع تایمز استفاده کنیم یک تیک کلاک تفاوت دارد
3.
4.
5. از فلگ های کامپایلر استفاده کنیم تا اپتیمایزیشن را غیر فعال کنیم. -0o
6. از تابع `pthread_setaffinity_np` استفاده کنیم.
7. می توانیم اول با calloc مقدار دهی کنیم و برنامه جوری باشد که صرفا لوپ را بررسی کند.