**Ⅰ. Scheduling Policy Goal**

**Minimize response time**

- Elapsed time[[1]](#footnote-1) to do on an operation (or job)

- Response time is what the user see

**Maximize throughput[[2]](#footnote-2)**

- Operations (or jobs) per second[[3]](#footnote-3)

- Minimize overhead ( context switching )

- Efficient use of system resources[[4]](#footnote-4) ( CPU, disk, memory etc )

**Fair**

- Share CPU among user in some equitable[[5]](#footnote-5) way

**Ⅱ. Scheduling Policies**

**1. FIFO ( FCFS[[6]](#footnote-6) / Run until done)**

- One program CPU until it completely finished.

- With strict uniprogramming, if have to wait for I/O, keep processor.

- **Pros** : simple ↔ **Cons** : short jobs get stuck behind jobs

**2. Round Robin**

- Add timer, and preempt CPU from long running jobs.

- After time slice, move thread to back of the queue.

- In some sense[[7]](#footnote-7), it’s fair ( each job gets equal shot at the cpu )

**Choose time slice**

- Too Big? Response time suffer[[8]](#footnote-8)s

- Too small ? Throughput suffers. Spend all your time context switching, none getting real work done.

- Time slice는 짧을수록 Fair(공평)하다. 10 ~100 milliseconds, so roughly 1% overhead due to time slicing.

- **Pros** : better for short jobs ↔ **Cons** : poor when jobs are same length

**3. STCF/SRTCF**

**STCF**

- Shortest time to completion first. Run whatever job has the least amount of stuff[[9]](#footnote-9) to do.

**STRCF**

- Shortest time remaining time to completion first.

- 현재 interrupt가 걸린 것을 ( Waiting 상태에서 Ready 상태로 간 것) 포함한다.

- Result is better average response time

- **Pros** : average response time optimal[[10]](#footnote-10) ↔ **Cons** : hard to predict to **future, unfair**

**Ⅲ. Comparison FIFO, Round Robin, STRCF**

**1. FIFO vs Round Robin**

Assumption :

- 10 jobs

- each take 100 seconds of CPU time

- Round Robin time slice of 1 second

- All start at the same time.

|  |  |  |
| --- | --- | --- |
| Job # | FIFO | Round Robin |
| 1 | 100 | 991 |
| 2 | 200 | 992 |
| 3 | 300 | 993 |
| 4 | 400 | 994 |
| 5 | 500 | 995 |
| 6 | 600 | 996 |
| 7 | 700 | 997 |
| 8 | 800 | 998 |
| 9 | 900 | 999 |
| 10 | 1000 | 1000 |
| Average | 550 | 995.5 |

**- If jobs are same length, average time is FIFO better than Round Robin**

**2. Same Length Jobs**

- If all jobs are the same length, SRTCF become the same as FIFO

- FIFO is as good as you can do if all jobs are the same length

**3. Vary[[11]](#footnote-11)ing Length Jobs**

- SRTCF ( and Round Robin ) : short jobs don’t get stuck behind long jobs

**4. FIFO vs Round Robin vs SRTCF**

Assumption

- Job A & B : Both CPU bound, run for week

- Job C : I/O bound, loop, 1ms of CPU, 10ms of disk I/O

**FIFO**

- once A or B get in, Keep CPU for two week.

- C를 먼저 한다면(C에는 기간이 없음으로 계속 수행한다.)

**Round Robin (100ms time slice )**

- Only get 5% disk utilization

**Round Robin (1ms time slice )**

- Get nearly 90% disk utilization

- Almost as good as C alone.

- A or B by all that much : they still get 90% of the CPU

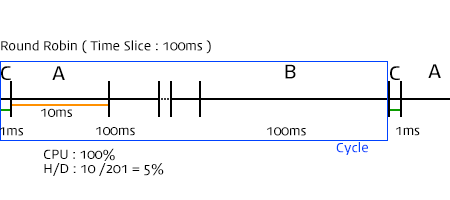
**SRTCF**

- No needless preemption

- Run C as soon as possible, run either A or B completion

- A downside(단점) to SRTCF is that it can lead to starvation[[12]](#footnote-12)

- Lots of short jobs can keep long jobs from making any progress

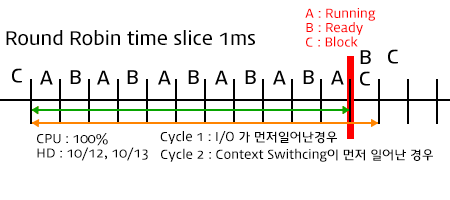


C→ Ready

Timeout ( B ready)

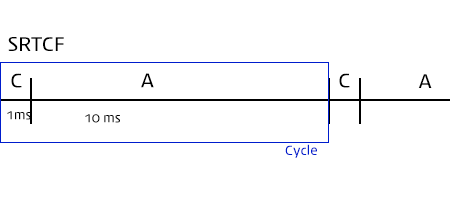
Timeout ( A ready)

I/O Blocked



해당 지점에서 I/O가 먼저 일어나는 지, Context Switch이 먼저 일어나는 지에 대한 순서는 원래 존재 한다.

두 가지 cycle에 대하여 다 파악할 수 있으며, 어떤 방식으로든 75%~85%사이의 Disk활용도를 얻을 수 있다.



I/O Complete

I/O Interrupt 로 CPU를 뺏는다

CPU preemption

I/O Request

H/D = 10/11 ≒ 90%이상

Shortest remaining 임으로 A만 계속 실행(**Unfair**)

Overhaed를 고려하면 10/11보다는 작을 것이다.

**Ⅳ. Knowledge of Future**

**1. Problem : STCF/SRTCF requires knowledge of the future**

**Step ①** Some system ask the user

- When you submit a job like a compile, have to say how long it will take

설명) Job을 컴파일 시킬 때 얼마나 걸리는지 시스템이 물어본다.

**Step ②** To stop cheating

- if your job takes more than what you said, system kills your job.

설명) 사람들이 서로 서로 속여서 빨리 하려고 함으로 자신의 시간보다 짧게 말을 한 경우에 Job을 가장 마지막으로 보내 버린다.

**Step ③** Start all over.

- Hard to predict resources usage in advance[[13]](#footnote-13)

설명) 따라서 사람들은 10이 걸린다면 12로 시스템에 알려주어야만 한다. 하지만 미래를 예측하는 것은 굉장히 어렵다.

**2. Multilevel Feedback**

- **Use past to predict future**

- Program behavior is regular, most of the time.

**Adaptive policies**

- Change policy based on past behavior.

- Used in CPU scheduling in virtual memory, in file system

설명) 과거의 행동을 통하여 정책을 변경한다.

**Multi-level feedback queues**

- Multiple queues, each with different priority

- OS does round robin at each priority level – run highest priority jobs first; once those finish, run next highest priority

- Round robin time slice increases exponentially[[14]](#footnote-14) at lower priorities

Timeout

Long Job

이렇게 가면 Short Job

**Adjust each job’s priority as follows(details vary)**

**Step ①** Job starts in highest priority queue.

설명) 최초에 들어온 것은 어떤 상태인지 모름으로 항상 우대한다.

**Step ②** If timeout expires, drop on level

설명) Timeout이 발생되어 졌다면 다음 level로 내린다.

**Step ③** If timeout doesn’t expire, push up one level(or back to top)

설명) Timeout이 발생되지 않으면 위로 올린다

하나의 Process도 I/O (short) → CPU(long) → I/O (short)로 바뀔 수 있다.

Problem : It is still unfair : long running jobs may never get the CPU

→ Countermeasure[[15]](#footnote-15) : 적당한 주기로 가장 밑에 있는 것을 위로 올려서 실행시킨다. ( 적당한 주기라는 것은 시스템상 올바르지 않다. )

**3. Lottery Scheduling**

- Give every job some number of lottery tickets, and on each time slice, randomly pick a winner ticket. On average, CPU time is proportional [[16]](#footnote-16)to # of tickets given to each job.

Short job : 10 tickets / Long job : 1 ticket

|  |  |  |
| --- | --- | --- |
| # short jobs / long jobs | % of CPU each  short job gets | % of CPU each long job gets |
| 1/1 | 10/11 : 91% | 1/11 : 9% |
| 0/2 | NA | 1/2 : 50% |
| 2/0 | 10/20 : 50% | NA |
| 10/1 | 10/101 : 10% | 1/101 : 1% |
| 1/10 | 10/20 : 50% | 1/20 : 5% |

설명) 각 Thread 가 복권을 가지고 있는데 복권은 최소한 1장 이상 가지고 있어야 한다.

최소 1장이라는 조건은 선택 시 0%가 되는 것을 막아준다.

Job 시작 시 Start라고 판단하고 복권을 많이 주고 시작한다.

timeout이 될 때마다 복권을 가져간다.

I/o block이 될 때마다 복권을 준다. Max값을 지정하고 그 이상은 주지 않는다.

1. elapsed time =경과시간 [↑](#footnote-ref-1)
2. throughput = 생산성 [↑](#footnote-ref-2)
3. 단위 시간당 몇 개의 Operation(or job)을 실행하는가 [↑](#footnote-ref-3)
4. 자원의 효율적 사용 [↑](#footnote-ref-4)
5. equitable = 공정한, 공평한 [↑](#footnote-ref-5)
6. First come first service [↑](#footnote-ref-6)
7. In some sense = 어떤 뜻으로는 [↑](#footnote-ref-7)
8. suffer = 시달리다 ( 길다 ) [↑](#footnote-ref-8)
9. stuff = 양 [↑](#footnote-ref-9)
10. optimal = 최선의 [↑](#footnote-ref-10)
11. vary = 다르다. [↑](#footnote-ref-11)
12. starvation = 굶주림 [↑](#footnote-ref-12)
13. In advance = 사전에 [↑](#footnote-ref-13)
14. Exponentially = 지수 승 [↑](#footnote-ref-14)
15. Countermeasure = 대책 [↑](#footnote-ref-15)
16. Proportional = 비례하는 [↑](#footnote-ref-16)