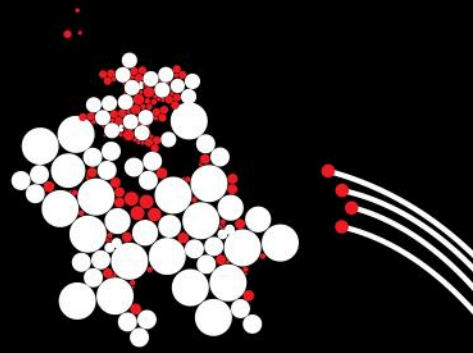
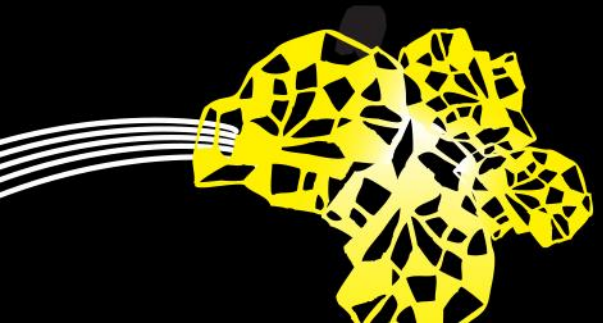
UNIVERSITY OF TWENTE.



OPERATING SYSTEMS

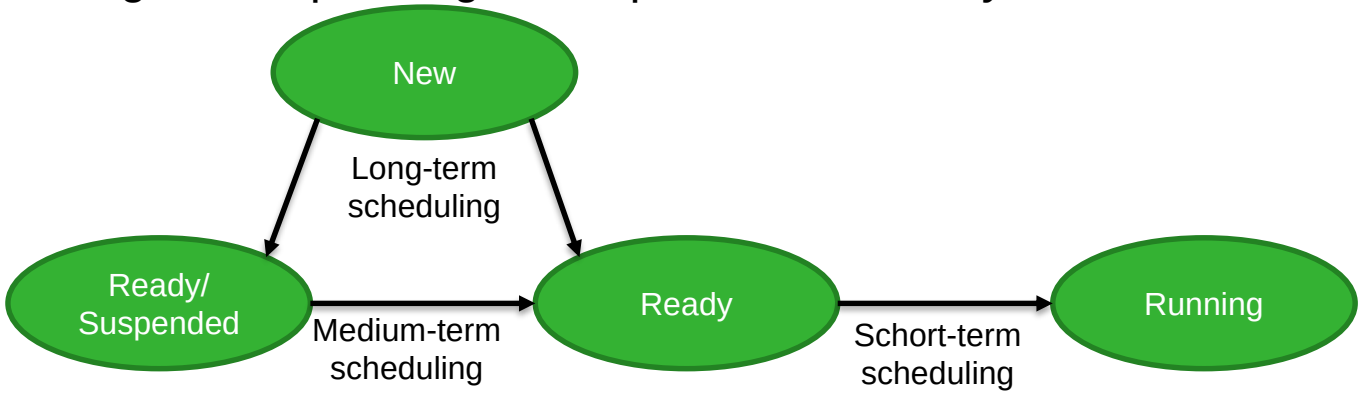
CPU SCHEDULING

ERIK TEWS <e.tews@utwente.nl>



TYPES OF SCHEDULING

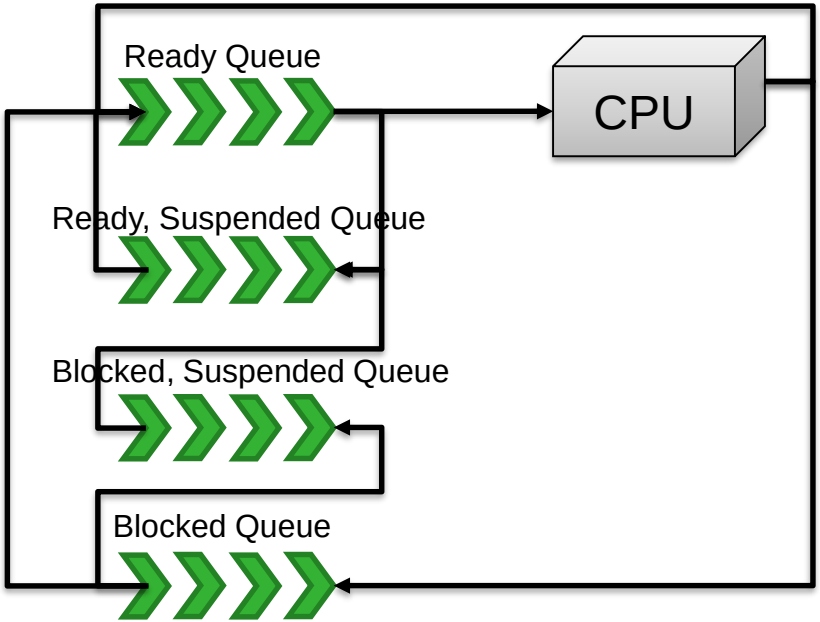
- Short-term: which runnable process to handle by which CPU
- Medium-term: which processes to swap in (challenge?)
- Long-term: which processes to accept in a batch environment
- I/O scheduling: which pending I/O request to handle by which I/O device



SCHEDULING

MANAGING QUEUES TO MINIMIZE DELAYS

- User oriented criteria: response time, deadlines, predictability
- System oriented criteria: throughput, resource utilization

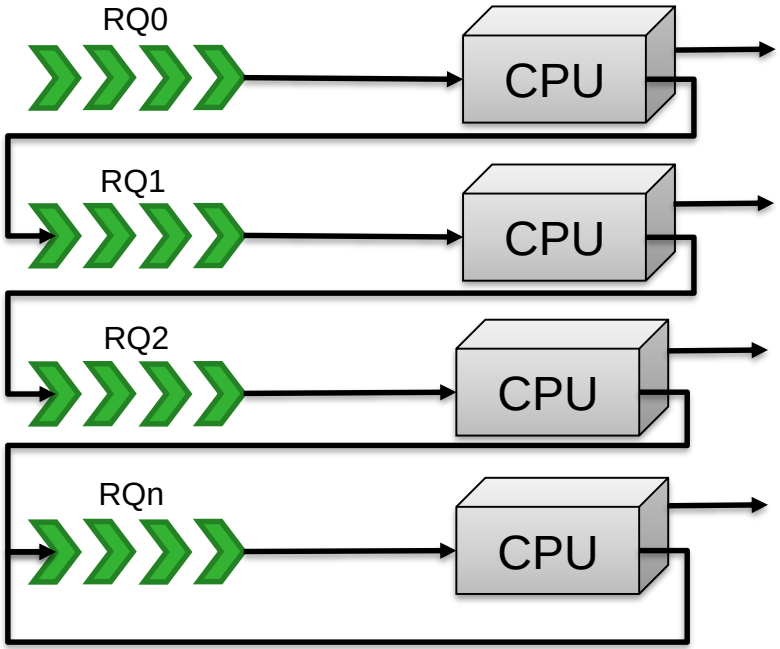


COMMON SCHEDULING ALGORITHMS

- First-Come-First-Served
- Round-Robin
- Feedback

MULTI-LEVEL FEEDBACK QUEUE

- Round Robin in RQi for 2^i time units
- Promote waiting processes
- Demote running processes



LINUX SCHEDULING (SECTION 10.3)

- Three levels
  - Real-time FIFO, pre-empted only by higher priority RT FIFO
  - Round robin RT, pre-empted by clock after quantum expiry
  - Normal scheduling of tasks using the Completely Fair Scheduler

LOOP

- gcc –c loop.c –o loop.o
- Write a test program

loop.h

```
#ifndef _loop_h
#define _loop_h 1
extern void loop(int N) ;
#endif
```

loop.c

```
#include "loop.h"

#define M 1690

/* Burn about N * 10 ms CPU time */
void loop(int N) {
    int i, j, k ;
    for(i = 0; i < N; i++) {
        for(j = 0; j < M; j++) {
            for(k = 0; k < M; k++) {
            }
        }
    }
}
```

SchedXY

- gcc –o RR80 loop.o
  -DX=SCHED_RR
  -DY=80 SchedXY.c
- sudo ./RR80&
- sudo ./RR80&
- top

```
int main(int argc, char *argv[]) {
    pid_t pid = getpid();
    struct sched_param param;
    param.sched_priority=Y;
    if( sched_setscheduler(pid,
        X, &param) != 0 ) {
        printf("cannot setscheduler\n");
    } else {
        loop(100);
    };
    return 0;
}
```

ThreadSched

- Output?
- gcc ThreadSched.c -lpthread
- ./a.out

- ./a.out xx

```
#define N 8
#define M 1000000

void *tproc(void *ptr) {
    int k, i = *((int *) ptr);
    int bgn = sched_getcpu();
    printf("thread %d on CPU %d\n",
        i,bgn);
    for(k=0;k<M;k++) {
        int now = sched_getcpu();
        if( bgn != now ) {
            printf("thread %d to CPU %d\n",
                i,now);
            break;
        }
        sched_yield();
    }
    pthread_exit(0);
}
```

NICE

- Output?
- gcc Nice.c loop.o
- ./a.out >junk&
- top
- ./a.out
- ./a.out xx

```
int main(int argc, char *argv[]) {
    int p, q, r;
    cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(1, &cpuset);
    sched_setaffinity(parent,
        sizeof(cpu_set_t), &cpuset);
    for( p = 0; p < P; p++ ) {
        for( q = 0; q < Q; q++ ) {
            pid_t child = fork();
            if (child == 0) {
                setpriority(PRIO_PROCESS,
                    getpid(), p) ;
                for(r = 0; r < R; r++) {
                    loop(100);
                }
                exit(0) ;
            }
        }
    }
}
```

SUMMARY

- Maximizing resource usage, while minimizing delays
- Decisions
  - Long term: admission of processes
  - Medium term: swapping
  - Short term: CPU assignment to ready process
- Criteria
  - Response time: users
  - Throughput: system