# CS302
# Operating System
# Lab 4

# Thread and Schedule

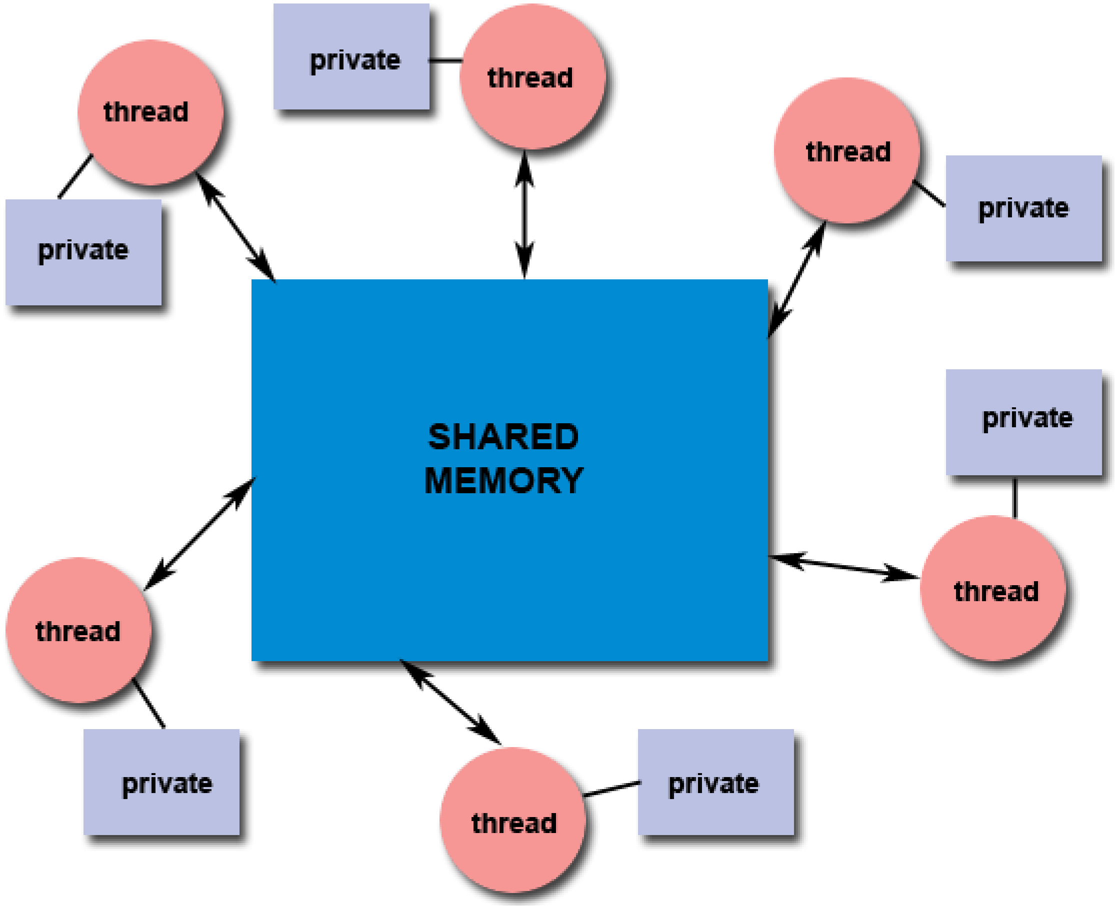
ZENG Xinxun, ZHANG Shiqi

# Outline

- Thread
  - Why multithreading?
  - Thread libraries
  - Practice
  - Threading issue
- Schedule
  - Why scheduling?
  - Three level scheduling
- Assignment: Scheduler

# Why multithreading?

- Responsiveness
- Scalability
- Resource sharing
- Economy




Picture from: https://computing.llnl.gov/tutorials/pthreads/

# Threading vs Processing

| Platform | fork() | | | pthread_create() | | |
|---|---|---|---|---|---|---|
| | real | user | sys | real | user | sys |
| **Intel 2.6 GHz Xeon E5-2670 (16 cores/node)** | 8.1 | 0.1 | 2.9 | 0.9 | 0.2 | 0.3 |
| **Intel 2.8 GHz Xeon 5660 (12 cores/node)** | 4.4 | 0.4 | 4.3 | 0.7 | 0.2 | 0.5 |
| **AMD 2.3 GHz Opteron (16 cores/node)** | 12.5 | 1.0 | 12.5 | 1.2 | 0.2 | 1.3 |
| **AMD 2.4 GHz Opteron (8 cores/node)** | 17.6 | 2.2 | 15.7 | 1.4 | 0.3 | 1.3 |
| **IBM 4.0 GHz POWER6 (8 cpus/node)** | 9.5 | 0.6 | 8.8 | 1.6 | 0.1 | 0.4 |
| **IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)** | 64.2 | 30.7 | 27.6 | 1.7 | 0.6 | 1.1 |
| **IBM 1.5 GHz POWER4 (8 cpus/node)** | 104.5 | 48.6 | 47.2 | 2.1 | 1.0 | 1.5 |
| **INTEL 2.4 GHz Xeon (2 cpus/node)** | 54.9 | 1.5 | 20.8 | 1.6 | 0.7 | 0.9 |
| **INTEL 1.4 GHz Itanium2 (4 cpus/node)** | 54.5 | 1.1 | 22.2 | 2.0 | 1.2 | 0.6 |

Source fork_vs_thread.txt

Picture from: https://computing.llnl.gov/tutorials/pthreads/

Time utility of 50,000 process/pthread creation

# Thread libraries

- Three main libraries:
  - POSIX Pthread
  - Windows thread
  - Java thread

# POSIX Pthread

- Compile flag: -pthread   e.g.  gcc thread.c –o thread –pthread
- API
  - pthread_create
  - pthread_exit
  - pthread_cancel
  - pthread_join
  - pthread_detach

```c
struct msg{
  int id;
  char* word;
};

void PrintHello(struct msg* arg)
{
  printf("%s%d!\n", arg->word, arg->id);
  pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
  pthread_t threads[NUM_THREADS];
  struct msg arg[NUM_THREADS];
  char* word = "Hello World! It's me, thread #";
  for(int t=0;t<NUM_THREADS;t++){
    printf("In main: creating thread %d\n", t);
    arg[t] = (struct msg){t, word};
    pthread_create(&threads[t], NULL, &PrintHello, &arg[t]);
  }
  for(int t=0;t<NUM_THREADS;t++)
    pthread_join(threads[t], NULL);
  printf("\nall threads finish.\n");
  /* Last thing that main() should do */
  pthread_exit(NULL);
}
```

# Implicit Threading

- Thread pool
  - A management pattern of threads.
- OpenMP
  - Threading automatically by compiler.
  - Try: gcc openmp1.c -o openmp1 –fopenmp
  - Try: gcc openmp2.c -o openmp2 -fopenmp

```c
int main (int argc, char *argv[])
{
    printf("Number of threads = %d\n", omp_get_num_threads());
    #pragma omp parallel
    {
     printf("I am thread %d\n", omp_get_thread_num());
    }
}
```

```c
int main (int argc, char *argv[])
{
    int a[80],b[80],c[80];
    for(int i=0;i<80;++i)
    {
        a[i] = i;
        b[i] = 2*i;
    }

    #pragma omp parallel for
    for (int i = 0; i < 80; ++i)
    {
        c[i] = a[i] + b[i];
        printf("i=%d, c[i]=%d\n", i, c[i]);
    }
}
```

# Practice

- Write a multithreaded program, user input a series of number, calculates three statistical values in different thread, print the result in given order.
- Input: 90 81 78 95 79 72 85
- Output: average: 82<br>minimum: 72<br>maximum: 95

# Threading issue

- fork() and exec()
- Signal handing
- Thread cancellation
- Thread-local storage
- Scheduler activations

In Operating System Concept – 4.6

# Overview

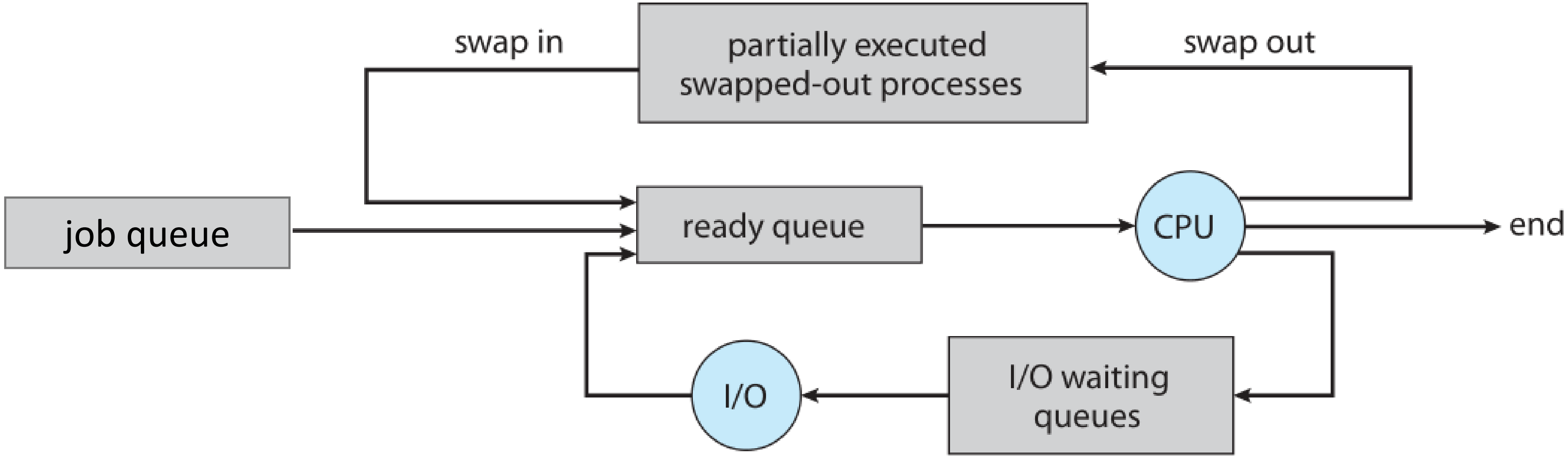
- Program, Process, Thread, Job
- OS Scheduler:
  - Long Term, Medium Term and Short Term
- Assignment: Scheduler

# Program, Process, Thread, Job

- **Program**: a group of instructions to carry out a specified task
  - stored on disk in some file.
- **Process**: an execution of certain program with its own address space
  - resources: CPU, memory address, disk, I/O etc.
- **Thread**: the unit of execution in CPU
- **Job**: a series of program submited to operating system for some goals.

# OS Scheduling

- **A job, after being submitted to operating system, will experience three kind of scheduling.**
  - Long Term Scheduler
  - Medium Term Scheduler
  - Short Term Scheduler




Source: Operating System Concepts 9th p114 Figure 3.7

# Background Terminology

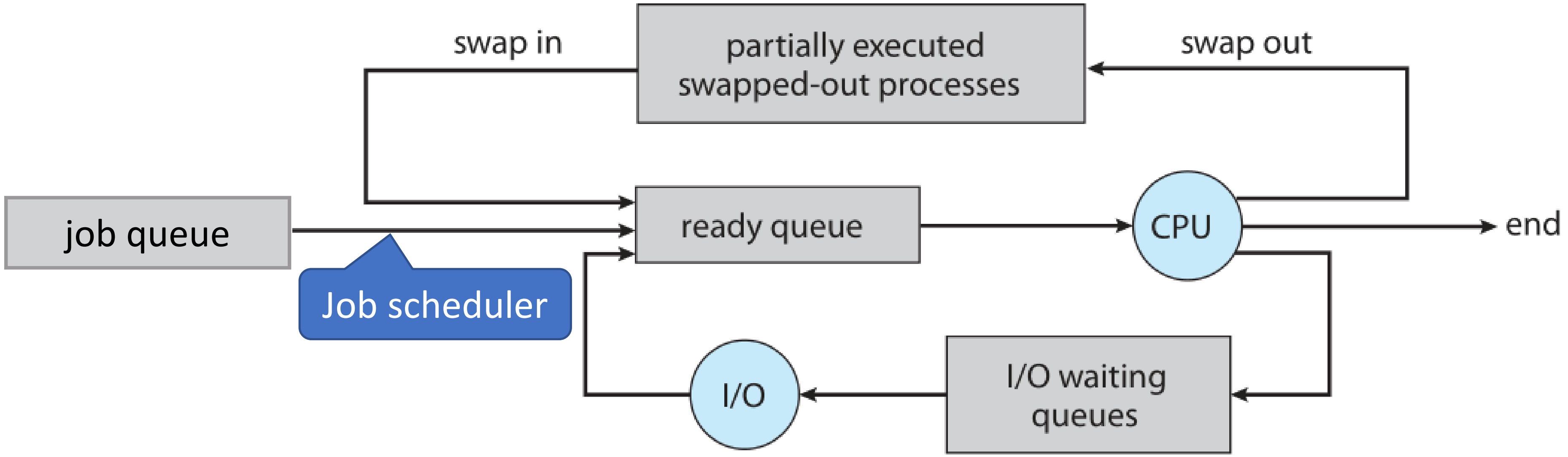
- **Degree of multiprogramming**
  - the number of processes in memory

- **I/O-bound process**
  - spends more time doing I/O than doing computations

- **CPU-bound process**
  - doing I/O infrequently but using more time on computations

# OS Scheduler

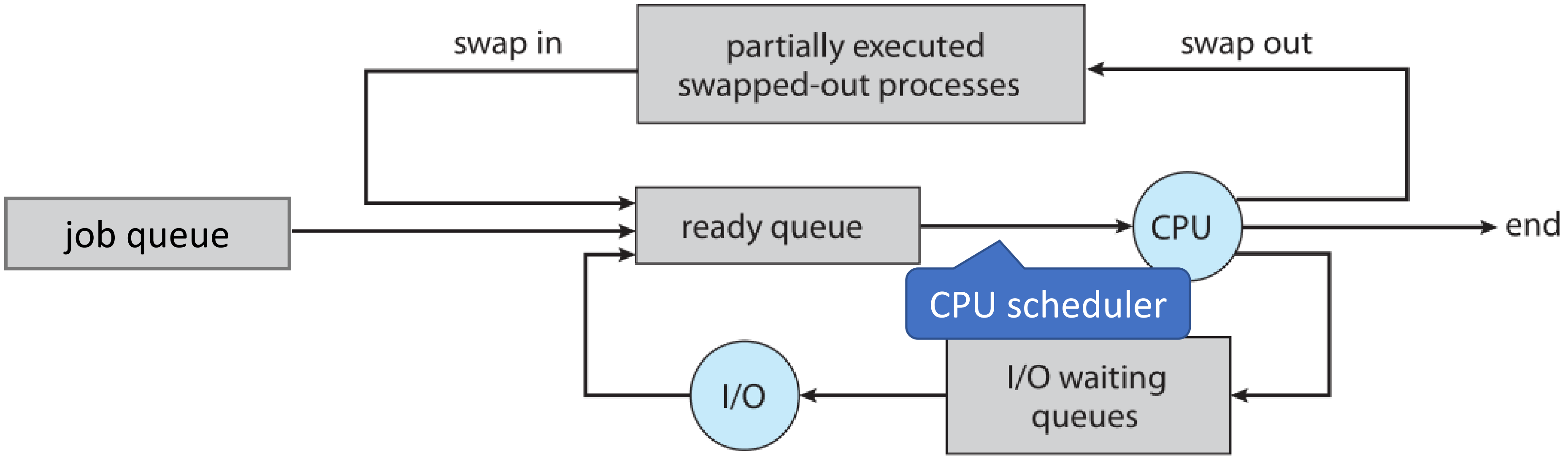
- Long Term Scheduler (**Job scheduler**)
  - Select processes from job queue(disk)
  - Load processes to ready queue(main memory)
  - Select a good **process mix** of *IO-bound* and *CPU-bound* processes
  - Control the **degree of multiprogramming**




Source: Operating System Concepts 9th p114 Figure 3.7

# OS Scheduler

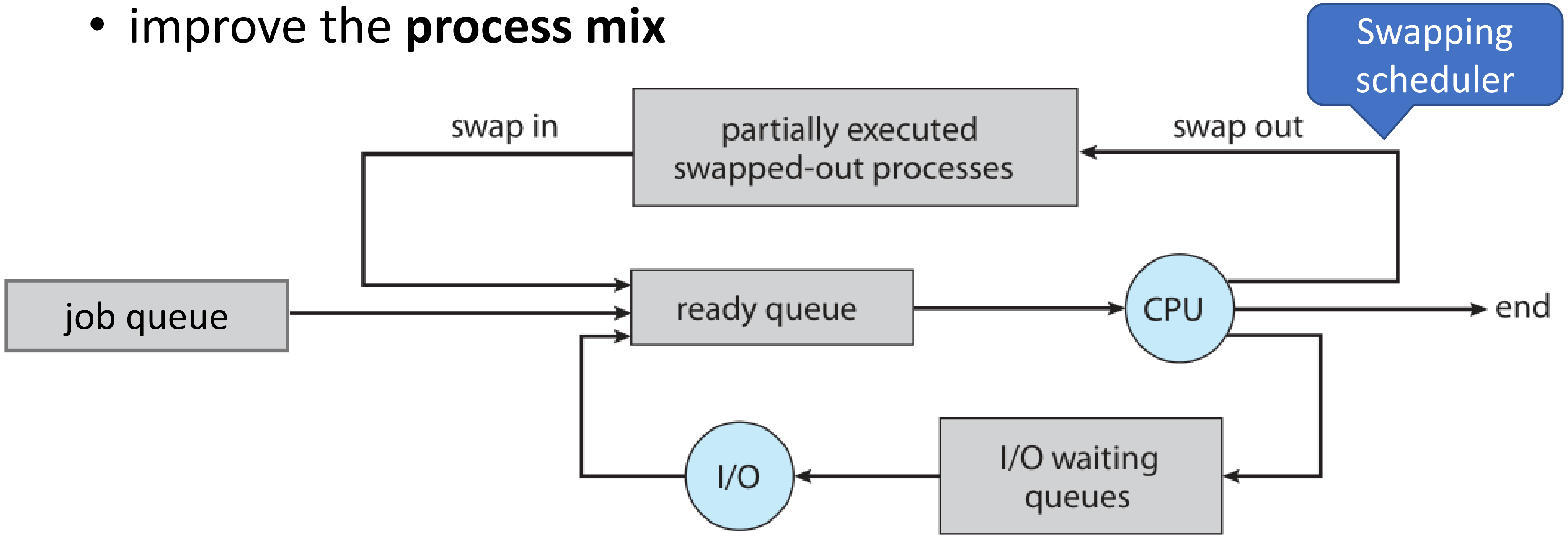
- Short Term Scheduler (**CPU scheduler**)
  - Select one process from ready queue
  - Execute it on CPU
  - Scheduling Algorithms: SJF, Round-Robin, Priority




Source: Operating System Concepts 9th p114 Figure 3.7

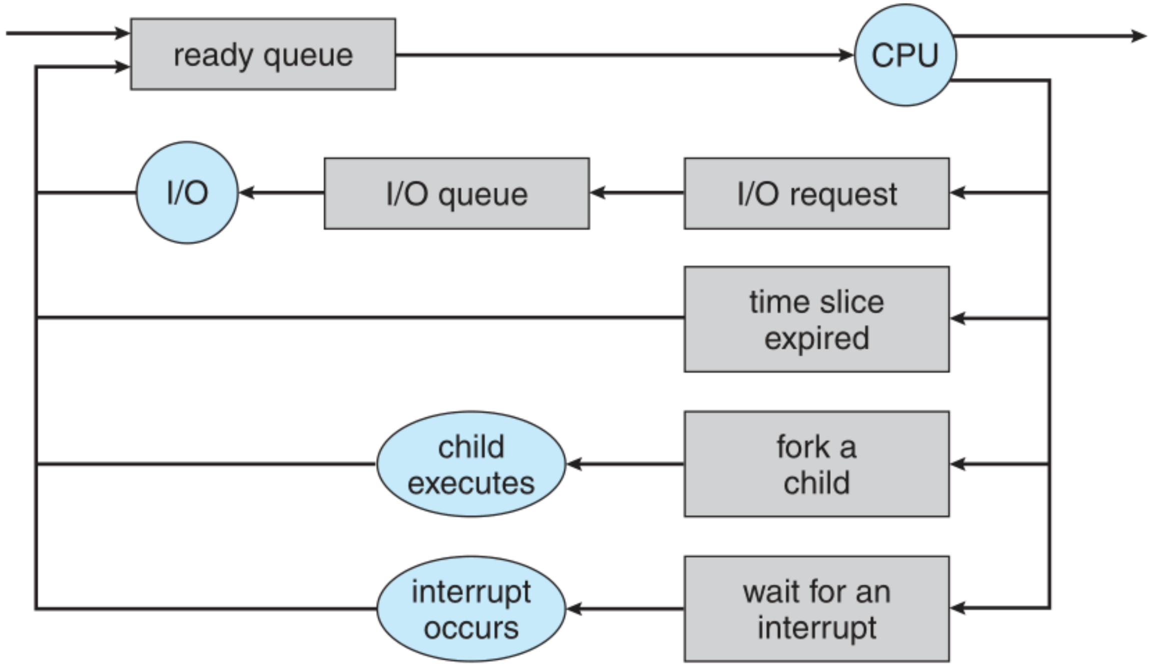
# OS Scheduler

- Medium Term Scheduler(**Swapping scheduler**)
  - **swap** a process(IO-bound/CPU-bound) **out** of memory
  - **swap** the process back **into** memory
  - reduce the **degree of multiprogramming**
  - improve the **process mix**




Source: Operating System Concepts 9th p114 Figure 3.7

# Where a process go after execution?




- normal terminate
- issue an I/O request and then be placed in an I/O queue
- create a new child process and wait for the child's termination
- remove from CPU due to an interrupt and back in the ready queue

Source: Operating System Concepts 9th p112 Figure 3.6

# OS Scheduler Comparison

| Long-Term Scheduler | Short-Term Scheduler | Medium-Term Scheduler |
|---|---|---|
| It is a job scheduler | It is a CPU scheduler | It is a process swapping scheduler |
| Speed is **lesser** than short term scheduler | Speed is **fastest** among other two | Speed is in **between** both short and long term scheduler |
| It **controls** the degree of multiprogramming | It provides **lesser control** over degree of multiprogramming | It **reduces** the degree of multiprogramming |
| It selects processes from **pool(disk)** and loads them into **memory** for execution | It selects those processes which are **ready to execute** | It can **re-introduce** the process into memory and execution can be continued |

https://www.tutorialspoint.com/operating_system/os_process_scheduling.htm

# Assignment

- You are asked to finish a simple CPU scheduler.
- One can add, remove, query jobs, the scheduler can schedule the job according the requirements.

# Strategy of CPU scheduler

- Nonpreemptive
- Preemptive
  - Priority
  - Shortest job first
  - Round-Robin

# Background

- Strategy: Priority based round-robin, time slice: 100ms
- Job Status: READY, RUNNING, DONE
- Priority Level: 0, 1, 2, 3 (from low to high)
- Initial priority vs Current priority

# Assignment

- File organization:
  - **sample.c:** a sample job which runs more than 100ms
  - **scheduler.c:** the main part of this scheduling system
  - **enq.c:** operations about job enqueue
  - **deq.c:** operations about job dequeue
  - **stat.c:** operations about printing status
  - **job.h:** relevant data structures for job

# Scheduler.c

- Function:
  - Create a process for a new job. Set its state to READY and put it in the waiting queue.
  - A job will dequeue when receive dequeue request, and the relevant data structure will be cleared.
    - If the job is currently running, it will first stop running and then dequeue.
  - If status request, it outputs information about the currently running job and all jobs in the waiting queue

# Enq.c

- Send a enqueue request to the scheduler and submit the job for running.
- Scheduler
  - assigns a unique jid to each job;
  - create a process for it and set its status to READY;
  - put this process into the waiting queue.

```
enq [-p num] e_file args
```

-p num： optional, assign an initial priority
e_file: the name of the executable file
args： the args of the executable file

# Deq.c

- Send a dequeue request to the scheduler

```
deq jid
```

jid： jid that need dequeue

# stats.c

- Print out the following information of current running jobs and jobs in waiting queue on standard output:
  - pid
  - user name
  - execution time
  - waiting time
  - create time
  - job status

# Job.h

- Define the data struct of all things.

- Define the path of FIFO(use for communicate between all components). You need to create a fifo file in this path to let it work.

# Task

- **Modify** do_stat() in scheduler.c such that it can also output ***jobname***, ***current priority*** and ***default priority***

- **Implement schedule strategy**: job with highest priority runs first, if with same priority, the job waiting longest runs first
  - Priority add 1 after waiting for 100ms
  - Priority reset to initial priority after running

- **Run** this scheduler system
  - do enqueue, dequeue and status operations
  - answer the questions in report