schedule-demo.c

typedef struct {

char\* name;

int arrival;

int execution;

int priority;

} request\_t;

/\* List jobs in order of arrival \*/

request\_t requests[] = {

/\*name, arrival, execution, priority \*/

{ "P1", 0 , 30 , 1},

{ "P2", 0 , 20 , 2},

{ "P3", 0 , 20 , 3},

{ "P4", 10 , 10 , 4}};

#define NUMBER(x) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int n\_requests = NUMBER(requests);

void \_\_\_\_\_\_\_\_\_\_\_\_() {

for(int i = 0; i < n\_requests;i++) {

if( ! ! can\_run(i) )

if( process == -1 || requests[i].execution < requests[process].execution )

process = i;

}

}

typedef \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; // scheduler\_fn ?

typedef struct {

char\* shortname;

scheduler\_fn fn;

} named\_scheduler\_t;

named\_scheduler\_t schedulers[] = {

{rr", round\_robin},

{"fcfs", first\_come\_first\_served}, .. others ..

};

scheduler\_fn to\_sched(char\*name) {

for(int i = 0; i < NUMBER(schedulers);i++)

if(!strcmp(name, schedulers[i].shortname)) return schedulers[i].fn;

return NULL;

}

int main(int argc, char\*\*argv) {

char\* schedname = argc > 1 ? argv[argc -1 ] :"";

scheduler\_fn schedule = to\_sched(schedname);

while( remaining\_processes >0 ) {

schedule();

char\* running = process < 0 ? "-": requests[process].name;

printf("%3d:%s\n", time, running);

}

}

epolldemo.c

void one(int write\_fd) {

sleep(1); write(write\_fd, "B", 1); sleep(1); write(write\_fd, "C", 1);

close(write\_fd);

}

void two(int write\_fd) {

write(write\_fd, "A", 1); sleep(3); write(write\_fd, "D", 1); close(write\_fd);

}

void main() {

int epoll\_fd = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_( 10 /\*Ignored\*/ );

int \*\*fds = malloc(2 \* sizeof(int \*));

struct \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_event;

for (int i = 0; i < 2; i++)

{

fds[i] = malloc(2 \* sizeof(int));

pipe(fds[i]);

int read\_fd = fds[i][0];

int write\_fd = fds[i][1];

pid\_t pid = fork();

if (pid == 0) /\* child \*/

{

close(read\_fd);

if (i == 0) { one(write\_fd); }

else if (i == 1) { two(write\_fd); }

exit(0);

}

else /\* parent \*/ close(write\_fd);

event.events = \_\_\_\_\_;

event.data.fd = \_\_\_\_\_\_\_\_\_;

**// int epoll\_ctl (int epfd, int op, int fd, struct epoll\_event \*event);**

epoll\_ctl( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

}

while (1)

{

struct epoll\_event ev;

**// int epoll\_wait (int epfd, struct epoll\_event \*events, int maxevents, int timeout);**

epoll\_wait(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

char c;

ssize\_t bytes = read(ev.data.fd, &c, 1);

if (bytes == 1)

printf("Read: %c\n", c);

else

if(bytes == 0) epoll\_ctl(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

}

}

epolldemo.c

void one(int write\_fd) {

sleep(1); write(write\_fd, "B", 1); sleep(1); write(write\_fd, "C", 1);

close(write\_fd);

}

void two(int write\_fd) {

write(write\_fd, "A", 1); sleep(3); write(write\_fd, "D", 1); close(write\_fd);

}

void main() {

int epoll\_fd = epoll\_create(10 /\*Ignored\*/ );

int \*\*fds = malloc(2 \* sizeof(int \*));

struct epoll\_event event;

for (int i = 0; i < 2; i++)

{

fds[i] = malloc(2 \* sizeof(int));

pipe(fds[i]);

int read\_fd = fds[i][0];

int write\_fd = fds[i][1];

pid\_t pid = fork();

if (pid == 0) /\* child \*/

{

close(read\_fd);

if (i == 0) { one(write\_fd); }

else if (i == 1) { two(write\_fd); }

exit(0);

}

else /\* parent \*/ close(write\_fd);

event.events = EPOLLIN;

event.data.fd = read\_fd;

epoll\_ctl( epoll\_fd, EPOLL\_CTL\_ADD, read\_fd, &event]);

}

/\* Read data via epoll\_wait() \*/

while (1)

{

struct epoll\_event ev;

epoll\_wait(epoll\_fd, &ev, 1, -1);

char c;

ssize\_t bytes = read(ev.data.fd, &c, 1);

if (bytes == 1)

printf("Read: %c\n", c);

else

epoll\_ctl(epoll\_fd, EPOLL\_CTL\_DEL, ev.data.fd, NULL);

}

}

typedef struct {

char\* name;

int arrival;

int execution;

int priority;

} request\_t;

/\* List jobs in order of arrival \*/

request\_t requests[] = {

/\*name, arrival, execution, priority \*/

{ "P1", 0 , 30 , 1},

{ "P2", 0 , 20 , 2},

{ "P3", 0 , 20 , 3},

{ "P4", 10 , 10 , 4}};

#define NUMBER(x) (sizeof(x) / sizeof(x[0]))

int n\_requests = NUMBER(requests);

void \_\_\_\_\_\_\_\_\_\_\_\_() {

for(int i = 0; i < n\_requests;i++) {

if( ! ! can\_run(i) )

if( process == -1 || requests[i].execution < requests[process].execution )

process = i;

}

}

typedef void (\*scheduler\_fn) ();

typedef struct {

char\* shortname;

scheduler\_fn fn;

} named\_scheduler\_t;

named\_scheduler\_t schedulers[] = {

{rr", round\_robin},

{"fcfs", first\_come\_first\_served}, ..

};

scheduler\_fn to\_sched(char\*name) {

for(int i = 0; i < NUMBER(schedulers);i++)

if(!strcmp(name, schedulers[i].shortname)) return schedulers[i].fn;

return NULL;

}

int main(int argc, char\*\*argv) {

char\* schedname = argc > 1 ? argv[argc -1 ] :"";

scheduler\_fn schedule = to\_sched(schedname);

while( remaining\_processes >0 ) {

schedule();

name = process < 0 ? "-": requests[process].name;

printf("%3d:%s\n", time, name);

}