**Assignment 10**

Operating Systems

 Submitted by:

|  |
| --- |
|  |
| Logo, company name  Description automatically generated |  |

Godala Sukumar Reddy(2020BTechCSE032)

Institute of Engineering and Technology (IET)

 JK Lakshmipat University, Jaipur

                                                               November 2022

#include <bits/stdc++.h>

using namespace std;

#define PROFILE\_BREAK 499

#define PAGES 1024

class PhysicalMemory {

   public:

    int memory[PAGES] = {};

    int last\_location = 0;

    int occupied\_size() {

        int occ\_size = 0;

        for (int i = 0; i < PAGES; i++) {

            if (memory[i] == 1) {

                occ\_size++;

            }

        }

        return occ\_size;

    }

    int remaining\_size() {

        return PAGES - occupied\_size();

    }

};

enum JobDistribution {

    LJ,

    SJ

};

enum ReplacementPolicy {

    BF,

    WF,

    FF,

    NF,

};

class JobDetails {

   public:

    int id = 0;

    int size = 0;

    int base = 0;

    int burst\_time = 0;

    int found\_size = 0;

    JobDetails(int \_id, int \_size, int \_base, int \_burst\_time, int \_found\_size) {

        this->id = \_id;

        this->size = \_size;

        this->base = \_base;

        this->burst\_time = \_burst\_time;

        this->found\_size = \_found\_size;

    }

    bool is\_executable() {

        return this->found\_size >= this->size;

    }

    void run\_one\_step(PhysicalMemory\* physical\_memory) {

        burst\_time--;

        if (burst\_time == 0) {

            for (int i = base; i < base + size; i++) {

                physical\_memory->memory[i] = 0;

            }

        }

    }

    bool is\_completed() {

        return burst\_time == 0;

    }

    void terminate(PhysicalMemory\* physical\_memory) {

        burst\_time = 0;

        for (int i = base; i < base + size; i++) {

            physical\_memory->memory[i] = 0;

        }

    }

};

class Job {

   public:

    static JobDetails\* generate\_job(int id, PhysicalMemory\* physical\_memory, JobDistribution job\_distribution, ReplacementPolicy replacement\_policy) {

        int job\_size = get\_size\_by\_distribution(job\_distribution);

        int base = 0;

        srand(time(NULL));

        int burst\_time = rand() % 10 + 1;

        int found\_size = 0;

        if (replacement\_policy == ReplacementPolicy::BF) {

            found\_size = 0;

            int best\_fit\_base = 9999;

            for (int i = 0; i < PAGES; i++) {

                if (physical\_memory->memory[i] == 0) {

                    found\_size++;

                    if (found\_size >= job\_size && found\_size < best\_fit\_base) {

                        best\_fit\_base = found\_size;

                        base = i - found\_size + 1;

                    }

                } else {

                    found\_size = 0;

                }

            }

        } else if (replacement\_policy == ReplacementPolicy::FF) {

            found\_size = 0;

            for (int i = 0; i < PAGES; i++) {

                if (physical\_memory->memory[i] == 0) {

                    found\_size++;

                    if (found\_size == job\_size) {

                        base = i - found\_size + 1;

                        break;

                    }

                } else {

                    found\_size = 0;

                }

            }

        } else if (replacement\_policy == ReplacementPolicy::NF) {

            found\_size = 0;

            for (int i = physical\_memory->last\_location; i < PAGES; i++) {

                if (physical\_memory->memory[i] == 0) {

                    found\_size++;

                    if (found\_size == job\_size) {

                        base = i - found\_size + 1;

                        physical\_memory->last\_location = base;

                        break;

                    }

                } else {

                    found\_size = 0;

                }

            }

        } else if (replacement\_policy == ReplacementPolicy::WF) {

            found\_size = 0;

            int best\_fit\_base = -9999;

            for (int i = 0; i < PAGES; i++) {

                if (physical\_memory->memory[i] == 0) {

                    found\_size++;

                    if (found\_size >= job\_size && found\_size > best\_fit\_base) {

                        best\_fit\_base = found\_size;

                        base = i - found\_size + 1;

                    }

                } else {

                    found\_size = 0;

                }

            }

        } else {

            cout << "Something went wrong!" << endl;

        }

        if (found\_size < job\_size) {

            found\_size = 0;

            base = 0;

        }

        for (int i = base; i < base + job\_size; i++) {

            physical\_memory->memory[i] = 1;

        }

        JobDetails\* job = new JobDetails(id, job\_size, base, burst\_time, found\_size);

        return job;

    }

   private:

    static int get\_size\_by\_distribution(JobDistribution job\_distribution) {

        srand(time(NULL));

        if (job\_distribution == JobDistribution::LJ) {

            return rand() % 525 + 500;

        } else {

            return rand() % 499 + 1;

        }

    }

};

JobDetails\* find\_job(vector<JobDetails\*> jobs) {

    srand(time(NULL));

    int index = rand() % jobs.size();

    return jobs[index];

}

JobDetails\* find\_running\_job(vector<JobDetails\*> jobs) {

    JobDetails\* job = find\_job(jobs);

    while (job->is\_completed()) {

        job = find\_job(jobs);

    }

    return job;

}

int main() {

    PhysicalMemory\* pm = new PhysicalMemory();

    vector<JobDetails\*> jobs;

    int id = 0;

    while (true) {

        JobDetails\* generated\_job = Job::generate\_job(id, pm, JobDistribution::SJ, ReplacementPolicy::BF);

        if (generated\_job->is\_executable()) {

            jobs.push\_back(generated\_job);

            id++;

        } else {

            int released = 0;

            while (generated\_job->size > pm->remaining\_size()) {

                JobDetails\* running\_job = find\_running\_job(jobs);

                released = running\_job->size;

                running\_job->terminate(pm);

            }

        }

        for (JobDetails\* job : jobs) {

            if (!job->is\_completed()) {

                job->run\_one\_step(pm);

                cout << "id: " << job->id << ", time left: " << job->burst\_time << ", size occupied: " << job->size << endl;

            }

        }

        sleep(1);

    }

    for (int i = 0; i < 100; i++) {

        cout << pm->memory[i] << "";

    }

    return 0;

}