پروژه سیستم عامل 2

امیر خلیلی

مسیح حاج سعیدی

ورودی:

هدف این پروژه پیاده سازی شبیه سازی روش صفحه بندی مدیریت حافظه در سیستم عامل است. ابتدا اندازه حافظه, اندازه صفحات و اندازه ی زمان بیشینه ی اجرا را به عنوان ورودی میدهیم. سپس ورودی(به اسم sample1.txt) دیگر که در آن تعداد فرآیند ها, شماره یا اسم هر فرآیند, زمان شروع و زمان پایان هر فرآیند, تعداد حافظه و حجم هر حافظه ی مورد نیاز هر فرآیند را توسط تابع read_file میخوانیم.

کلاس process:

این کلاس, کلاس فرآیند است که شامل زمان شروع, زمان پایان, تعداد حافظه های مورد نیاز و ظرفیت حافظه های مورد نیاز است.

کلاس main_memory:

این کلاس مجوعه ی فرآیند ها را شامل میشود.

الگوریتم صفحه بندی:

در این قسمت تابع algorithm که پیاده سازی شبیه سازی الگوریتم صفحه بندی است را بررسی میکنیم.

ابتدا فرآیند ها را با توجه به زمان شروع آن ها در Priority Queue به اسم pq قرار میدهیم. یعنی فرآیند هایی که که زمان شروع کمتری دارند, اولویت بیشتری دارند.سپس فرآیند برای که اولویت بیشتری دارند را از صف اولویت بر میداریم. اگر تفاوت زمان پایان و زمان شروع فرآیند از ماکسیمم زمان اجرا بیشتر شد, آن فرآیند را در کل حذف میکنیم.

```
if (newP.finishTime - newP.startTime > max_time)
{
    output << "Process " << newP.name << " cant fit, we cant use this process because its time is bigger than maximum time" << endl;
    output << "it is deleted from the queue." << endl;
    continue;
}
```

اگر تفاوت زمان پایان و زمان شروع کمتر بود, آن فرآیند را در حافظه قرار میدهیم.

سپس به قسمت replacement algorithm می رسیم. در این قسمت اگر در حافظه فرایند ها از قبل وجود داشته باشند و بخواهیم فرآیند جدیدی را وارد کنیم که برای فرآیند جدید حافظه مورد نیاز وجود نداشت, از این الگوریتم استفاده میکنیم. شرط جایگزینی فرآیند قدیمی با فرآیند جدید این است که زمان کل اجرای فرآیند جدید از بیشینه زمان اجرای باقی مانده کمتر باشد.

عکس کد replacement algorithm میتوانید در زیر ببینید.

```
}

// replacment algo.
if (newP.finishTime - newP.startTime <= max_time && howMuchMemory(memory_map) < newP.total_memory())
{

    bool f1 = false;
    for (int i = 0; i < m.mem.size(); i++)
    {
        if (m.mem[i].name != newP.name && m.mem[i].total_memory() + howMuchMemory(memory_map) >= newP.total_memory())
        {
            output << "Process " << newP.name << "(replacement algo.)  -> we chose the best place(memory pages) for this process." << endl;
            f1 = true; // process + free >= new process (replace with 1 process)
            //cout << m.mem[i].total_memory()  << howMuchMemory(memory_map) << " " << newP.total_memory();
            for (int j = 0; j < memory_map.size(); j++)
            {
                if (get<1>(memory_map[j]) == "Process " + to_string(m.mem[i].name)) {
                    get<0>(memory_map[j]) = 0;
                    get<1>(memory_map[j]) = "";
                    get<2>(memory_map[j]) = "";
                }
            }
            break;
        }
    }
```

```
    if (!f1) // new process (replace with more than 1 process)
    {
        output << "Process " << newP.name << "(replacement algo.)  -> we chose the best place(memory pages) for this process" << endl << "\t\t\t\t\t" << "(mu
        vector<int> name_of_process_that_can_be_deleted;
        int total = howMuchMemory(memory_map);
        for (int i = 0; i < m.mem.size(); i++)
        {
            if (m.mem[i].name != newP.name)
            {
                name_of_process_that_can_be_deleted.push_back(m.mem[i].name);
                total += m.mem[i].total_memory();
            }
            if (total >= newP.total_memory())
            {
                for (int j = 0; j < memory_map.size(); j++)
                {
                    for (int z = 0; z < name_of_process_that_can_be_deleted.size(); z++)
                    {
                        if (get<1>(memory_map[j]) == "Process " + to_string(m.mem[z].name)) {
                            get<0>(memory_map[j]) = 0;
                            get<1>(memory_map[j]) = "";
                            get<2>(memory_map[j]) = "";
                        }
                    }
                }
                break;
            }
        }
    }
}
```

با توجه به داکیومنت پروژه الگوریتم replacement نیاز نبود یعنی ما می توانستیم فرآیند جدید را که زمان اجرای آن از بیشینه زمان اجرای باقی مانده بیشتر بود را وارد حافظه نکنیم.

نکته: علامت ستاره در صفحه ها نشان دهنده ی internal fragmentation است.

تکه تکه شدن داخلی را در این قسمت از کد بررسی میکنیم.

```
float number_of_page = (float)newP.total_memory() / (float)size_page; // for memory sizes that are not divisble by size of pages.

if (roundf(number_of_page) == number_of_page) // integer
{
    int count = 1;
    for (int i = 0; i < memory_map.size(); i++)
    {
        if (get<0>(memory_map[i]) == 0 && count <= number_of_page)
        {
            get<0>(memory_map[i]) = size_page;
            get<1>(memory_map[i]) = "Process " + to_string(newP.name);
            get<2>(memory_map[i]) = "Page " + to_string(count++);
        }
    }
}
else {// float
    int count = 1;
    int integer = floor(number_of_page);
    float fraction = (float)number_of_page - (float)floor(number_of_page);

    if (integer != 0)
    {
        for (int i = 0; i < memory_map.size(); i++)
        {
            if (get<0>(memory_map[i]) == 0 && count <= integer)
            {
                get<0>(memory_map[i]) = size_page;
                get<1>(memory_map[i]) = "Process " + to_string(newP.name);
                get<2>(memory_map[i]) = "Page " + to_string(count++);
            }
        }
    }
    for (int i = 0; i < memory_map.size(); i++) {
        if (get<0>(memory_map[i]) == 0) {
            get<0>(memory_map[i]) = size_page * fraction;
            get<1>(memory_map[i]) = "Process " + to_string(newP.name);
            get<2>(memory_map[i]) = "Page " + to_string(count++);
            break;
        }
    }
}
```