KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG MÁY TÍNH VÀ TT DỮ LIỆU

Hướng dẫn Lab 9.4 – Cấp phát bộ nhớ liên tục.

Cấp phát bộ nhớ là một trong những thành phần cốt lõi của một Hệ điều hành. Trong phương pháp cấp phát liên tục, một tiến trình sẽ được nạp vào một lỗ trống với một giải thuật thích hợp, Hệ điều hành cập nhật danh sách các lỗ trống để quản lý.

Mục tiêu	**Lý thuyết liên quan**	**Tài nguyên**
Cấp phát liên tục	Ch9.2 Contiguous Memory Allocation Ch9.2.2 Memory Allocation Ch9.2.3 Fragmentation	https://github.com/Trantin84/LAB_IntroOS (mã nguồn ví dụ). Sử dụng image Ubuntu 16 / 18
Chọn lỗ trống bằng First-fit, Best-fit và Worst-fit.		
Chống phân mảnh		

Yêu cầu sinh viên: Hiểu và thực thi các đoạn mã đã cung cấp. Áp dụng lý thuyết hiện thực các giải thuật cấp phát và chống phân mảnh.

Đánh giá sinh viên: Hỏi đáp các vấn đề lý thuyết. Kỹ năng thực hành. Bài tập.

Yêu cầu nộp bài: các tập tin mã nguồn .c và tập tin khả thực thi .out của các Yêu cầu trong buổi thực hành và Bài tập cuối hướng dẫn trong thời gian cho phép của giảng viên.

Preferences

[1] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, [2018], Operating System Concepts, 10th edition, John Wiley & Sons, New Jersey.

Programming Problems of Chapter 9.

Mô tả cấu trúc vùng nhớ giả lập.

Bộ nhớ sẽ là phân phối thành các vùng liên tục: hoặc là một tiến trình đang cư trú (iPID của tiến trình); hoặc là lổ trống (iPID = -1). Mỗi phân vùng liên tục được quản lý bằng Địa chỉ bắt đầu iBase và kích thước của nó iSize. Ngoài ra phân vùng có thể chứa tên tiến trình hay một số thông tin ghi chú.

- Dòng 20 cho phép bộ nhớ phân làm tối đa 100 vùng. Số phân vùng thực tế kiểm soát bằng biến toàn cục iHoleCount.

- Dòng 22 chứa biến đếm để đánh ID cho tiến trình mới vào. Giá trị này tăng dần cho mỗi tiến trình.

```
14      struct hole {
15        int iPID; //-1 unused
16        int iBase;
17        int iSize;
18        char sName[20];
        };
20      struct hole M[100];
21      int iHoleCount = 0;
22      int iPIDcount = 1000;
```

Khi khởi tạo, vùng nhớ được xem là lổ trống duy nhất, khả dụng và có kích thước được truyền vào qua lời gọi.

```
86      M[iHoleCount].iSize = atoi(argv[1]);
87      M[iHoleCount].iPID = -1;
88      M[iHoleCount].iBase = 0;
89      iHoleCount = 1;
```

Yêu cầu 1: Cấp phát vùng nhớ liên tục cho các tiến trình theo giải thuật First-fit.

Hướng dẫn: Tải về tập tin ***LAB_IntroOS/LAB_9/task4_1_memalloc.c***, biên dịch và thực thi.

- Khi một tiến trình yêu cầu kích thước iSizeNew, hệ thống (sử dựng First-fit) sẽ đi duyệt từ đầu bộ nhớ (dòng 30) và tìm lổ trống (dòng 31).

- Có 3 tình huống:

a) (Dòng 32) Lổ trống bé hơn: không thể cấp phát, vòng lặp tiếp tục.

b) (Dòng 33) Lổ trống đúng bằng kích thước tiến trình: nó được cấp vào đó, không có lổ trống mới tạo ra do không có dư kích thước. Tổng số phân vùng không thay đổi.
c) (Dòng 40) Lổ trống to hơn kích thước tiến trình: nó được cấp vào đó, có một lổ trống mới tạo ra từ phần kích thước dư còn lại sau khi cấp. Tổng số phân vùng tăng thêm 1.

```
30        for (int i = 0; i < iHoleCount; i++) {
31          if (M[i].iPID == -1) {
32            if (M[i].iSize < iSizeNew) continue;
33            else if (M[i].iSize == iSizeNew) {}
40            else if (M[i].iSize > iSizeNew) {}
```

Yêu cầu 2: Thu hồi bộ nhớ sau khi một tiến trình kết thúc.

Hướng dẫn:

- Khi một tiến trình kết thúc và rời khỏi hệ thống, vùng nhớ đã cấp được thu hồi, đánh dấu là "available".

```
64            M[i].iPID = -1;
```

- Tuy nhiên cần lưu ý rằng: nếu trước và/hoặc sau tiến trình kết thúc là một "lổ trống" thì chúng ta cần gộp các lổ trống liền kề vào nhau. Dưới đây liệt kê 4 tình huống có thể xảy ra trước và sau khi xoá một tiến trình.

Px	Pexit	Py
Px	Pexit	Hole
Hole	Pexit	Py
Hole	Pexit	Hole

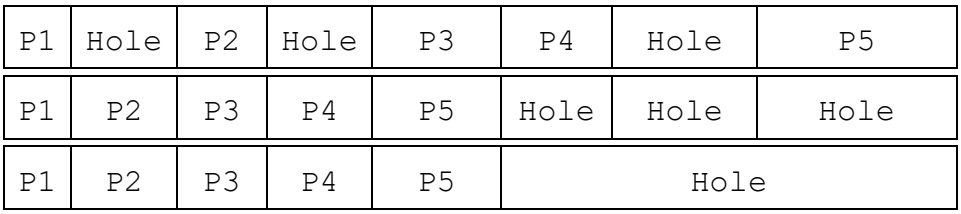
Px	Hole	Py
Px	Hole	
Hole		Py
Hole		

- Trong các tình huống gộp lổ trống, cần tính lại tổng số phân vùng iHoleCount, cũng như địa chỉ bắt đầu iBase và kích thước mới iSize của chúng.

Yêu cầu 3: Chống phân mảnh bộ nhớ.

Hướng dẫn: Bộ nhớ bị phân mảnh khi phải trải qua nhiều lần cấp phát và thu hồi, tạo ra vô số lổ trống rời rạc, mặc dù tổng dung lượng của các lổ trống này tương đối lớn, nhưng lại không thể dùng cho một tiến trình tiếp theo. Chống phân mảnh là việc dồn các tiến trình đang cư trú trong bộ nhớ về một phía, và vì thế, các lổ trống dồn về phía còn lại rồi hợp với nhau thành một lỗ trống

duy nhất. Sơ đồ dưới đây thể hiện: Trạng thái khi bị phân mảnh; Bước 1: dồn tiến trình về 1 phía và Bước 2: hợp nhất các lổ trống ở phía bên kia lại với nhau.




```
73
insert
code
and
debug

just
an
idea

not
yet
tested

good
luck
```

```
void * fCompact(void * param) {}
  int iReAlloc = 0;
  int iHoleCollect = 0;
  int iSizeCollect = 0;

  for (int i = 0; i < iHoleCount; i++) {
    if (M[i].iPID == -1){  //if meet a hole
      iReAlloc -= M[i].iSize;
      iHoleCollect++;
      iSizeCollect += M[i].iSize;  }
    else {   //if meet a Process
      M[i-iHoleCollect].iPID = M[i].iPID;
      M[i-iHoleCollect].iBase = M[i].iBase + iReAlloc;
      M[i-iHoleCollect].iSize = M[i].iSize;}

  iHoleCount -= iHoleCollect + 1;  //merge all holes
  M[iHoleCount-1].iPID = -1;
  M[iHoleCount-1].iBase = M[iHoleCount-2].iBase +
                          M[iHoleCount-2].iSize;
  M[iHoleCount-1].iSize = iSizeCollect;
```

Bài tập

1. Hoàn tất hàm và sử dụng để cấp phát vùng nhớ liên tục cho các tiến trình theo giải thuật Best-fit.
   Gợi ý: Lổ trống vừa vặn nhất chính là lổ trống nhỏ nhất trong tập hợp H "các lổ trống chứa được tiến trình". Với First-fit, lổ trống chứa được tiến trình tìm thấy đầu tiên được cấp ngay lập tức (và vì thế giải thuật này chạy nhanh nhất); thì với Best-fit, quá trình duyệt tiếp tục

cho đến lổ trống cuối cùng để hoàn chỉnh tập hợp H. Lưu ý thêm rằng, tập H có thể rỗng sau khi duyệt, khi đó, tiến trình không được cấp bộ nhớ.

Khi tìm thấy lổ trống Best-fit, cấp phát tiến trình vào đó, và phần bộ nhớ dư ra (nếu có) lại thành một lổ trống mới.

```
replace     int iFound = -1;
code        int iBestSize = -1;
and         for (int i = 0; i < iHoleCount; i++) {
debug         if (M[i].iPID == -1) {
                if (M[i].iSize < iSizeNew) continue;
                else if (iBestSize == -1 ||
just                      iBestSize > M[i].iSize) {
an                iFound = i;
idea              iBestSize = M[i].iSize;
                }
              }
good        }
luck        if(iFound != -1)  {//xử lý cấp phát, tham khảo Y/c 1
            else { //no hole found  }
```

2. Hoàn tất hàm và sử dụng để cấp phát vùng nhớ liên tục cho các tiến trình theo giải thuật Worst-fit.

   Gợi ý: Lổ trống lớn nhất trong tất cả lổ trống sẽ được tìm thấy sau khi duyệt toàn bộ. Nếu lổ trống này đủ chứa tiến trình thì tiến hành cấp phát.

```
replace     int iFound = -1;
code        int iBestSize = -1;
and         for (int i = 0; i < iHoleCount; i++) {
debug         if (M[i].iPID == -1) {
                if (M[i].iSize < iSizeNew) continue;
just            else if (iBestSize < M[i].iSize) {
an                iFound = i;
idea              iBestSize = M[i].iSize;
                }
good          }
luck        }
```

	`if(iFound != -1)  {//xử lý cấp phát, tham khảo Y/c 1` `else { //no hole found  }`

3. Tính tỉ lệ phân mảnh bộ nhớ để bổ sung vào chức năng "Thống kê".
   Gợi ý: Tỉ lệ phân mảnh = Tổng dung lượng các lổ trống / Tổng dung lượng bộ nhớ.
   - Khi hệ thống cấp phát không thành công, tỉ lệ này sẽ được tham khảo: nếu tỉ lệ này * kích thước bộ nhớ > kích thước tiến trình cấp thất bại thì việc chống phân mảnh sẽ giúp cho việc cấp phát lại thành công.

4. (SV tự học) Hiện thực Hàm **Randomize()** thể hiện xen kẽ 100 lần cấp phát và 100 lần thu hồi.
   Gợi ý: cần chỉnh sửa lại 2 hàm cấp phát và thu hồi.
   void * fAllocation(void * param);
   Kích thước tiến trình cần truyền trực tiếp qua param, thay vì hỏi người dùng nhập kích thước.

   void * fTerminate(void * param);
   ID tiến trình cần huỷ bỏ truyền trực tiếp qua param, thay vì hỏi người dùng nhập ID.

   main()
   - Gọi cấp phát 10 lần với kích thước ngẫu nhiên từ 1 đến MAX bộ nhớ.
   - Gọi huỷ bỏ 5 tiến trình với các ID xen kẽ. ID của tiến trình cấp đã gán tự động từ 1000, 1001, 1002, ….
   - Lặp đi lặp lại việc cấp và huỷ bổ sau một số lần sẽ tạo ra bộ nhớ phân mảnh nặng nề.