**Politechnika Warszawska**
**Wydział Elektroniki i Technik Informacyjnych**

# Systemy operacyjne

# Zarządzanie pamięcią
# Raport

**Zdający:**
**Jakub Sikora**

**Prowadzący:**
**mgr. inż. Aleksander Pruszkowski**

**Warszawa, 9 stycznia 2019**

# Spis treści

# 1. Treść zadania

## 1.1. Cel zadania

Domyślnie w systemie Minix algorytmem wyboru wolnego bloku z listy wolnych bloków, wykorzystywanym do realizacji funkcji systemowych FORK i EXEC, jest algorytm first fit, czyli wybierany jest pierwszy blok pamięci o wystarczającym rozmiarze z listy bloków wolnych.

Celem ćwiczenia jest zmiana domyślnego algorytmu przydziału pamięci w systemie Minix. Należy umożliwić wybór algorytmu wyboru bloku z listy bloków wolnych między standardowym first fit a tzw. algorytmem worst fit, czyli takim, w którym wybierany jest blok pamięci z listy wolnych bloków o największym rozmiarze.

## 1.2. Zadanie do zrealizowania

Należy zdefiniować dwie dodatkowe funkcje systemowe, identyfikowane stałymi HOLE_MAP oraz WORST_FIT.

### 1.2.1. HOLE_MAP

Funkcja systemowa HOLE_MAP powinna umożliwiać zdefiniowanie własnej funkcji o sygnaturze:

```
int hole_map(void *buffer, size_t nbytes)
```

która ma za zadanie zwrócić w buforze buffer o rozmiarze nbytes informacje o aktualnej zawartości listy wolnych bloków utrzymywanej przez moduł zarządzania pamięcią (MM). Struktura otrzymanej w buforze informacji powinna być następująca:

rozmiar1, adres1, rozmiar2, adres2, ..., 0

gdzie kolejne pary rozmiar, adres odpowiadają informacjom o kolejnych elementach listy wolnych bloków. Rozmiar 0 oznacza ostatni element listy. Elementy rozmiar i adres mają typ danych unsigned int (na poziomie modułu MM synonim tego typu o nazwie phys_clicks).

Funkcja hole_map ma zwracać przesłaną liczbę par rozmiar, adres. Należy zabezpieczyć się przed przepełnieniem zadanego jako argument wywołania bufora i wypełnić go tylko liczbą par mieszczących się w buforze dbając o zakończenie listy pozycją rozmiar=0.

### 1.2.2. WORST_FIT

unkcja systemowa WORST_FIT powinna umożliwiać wybór algorytmu wyboru elementu z listy wolnych bloków i zdefiniowanie własnej funkcji o sygnaturze:

```
int worst_fit(int w)
```

która dla w = 1 wymusza implementowany w ramach ćwiczenia algorytm przydziału worst fit, natomiast dla w = 0 uaktywnia z powrotem standardowy algorytm first fit. Wartością zwracaną powinno być zawsze 0.

# 2. Realizacja rozwiązania

## 2.1. Modyfikacja plików źródłowych MINIXa

Aby poprawnie zrealizować polecenie, należało zaimplementować algorytm `WORST_FIT`, zapewnić użytkownikowi systemu zmianę algorytmu oraz udostępnić dwa dodatkowe wywołania systemowe.

W pierwszej kolejności w `mm.h` zdefiniowałem dwie stałe symbolizujące wybrany algorytm alokacji pamięci.

```
#define FIRST_FIT 0
#define WORST_FIT 1
```

W tym samym pliku zadeklarowałem zmienną `mem_alg`, która przechowuje informacje o aktualnie wybranym sposobie alokacji pamięci.

Algorytm worst fit zaimplementowałem w pliku `alloc.c` w funkcji `alloc_mem`. Ciało zrealizowanej funkcji `alloc_mem`:

```
/*===========================================================*
                            alloc_mem
*===========================================================*/
PUBLIC phys_clicks alloc_mem(clicks)
phys_clicks clicks;
{

register struct hole *hp, *prev_ptr, *max_ptr, *max_prev;
phys_clicks old_base;

if(mem_alg == FIRST_FIT){
    do {
        hp = hole_head;
        while (hp != NIL_HOLE && hp->h_base < swap_base) {
            if (hp->h_len >= clicks) {
                old_base = hp->h_base;
                hp->h_base += clicks;
                hp->h_len -= clicks;

                if (hp->h_len == 0) del_slot(prev_ptr, hp);
                return(old_base);
            }

            prev_ptr = hp;
            hp = hp->h_next;
        }
  } while (swap_out());
    return(NO_MEM);
}

if(mem_alg == WORST_FIT){
    do {
        hp = hole_head;
        max_ptr = hole_head;
```

```
            max_prev = NIL_HOLE;

            while (hp != NIL_HOLE && hp->h_base < swap_base) {
                if(hp->h_len > max_ptr->h_len) {
                max_ptr = hp;
                max_prev = prev_ptr;
            }
            prev_ptr = hp;
            hp = hp->h_next;
        }
        if(max_ptr->h_len >= clicks) {
            old_base = max_ptr->h_base;
            max_ptr->h_base += clicks;
            max_ptr->h_len -= clicks;
            if(max_ptr->h_len == 0) del_slot(max_prev, max_ptr);
           return(old_base);
         }
       } while (swap_out());
     }
    }
```

W tym samym pliku zdefiniowałem dwie funkcje realizujące funkcjonalności nowych wywołań systemowych.

Wywołanie `HOLE_MAP` realizuje funkcja `do_hole_map()`.

```
/*===========================================================*
                do_hole_map
*===========================================================*/
PUBLIC int do_hole_map(void)
 {
    struct hole *temp;
    int i;
    int a = 0;
    char *buffer = mm_in.m1_p1;
    size_t nbytes = (size_t)mm_in.m1_i1;
    temp = hole_head;
    for(i = 0;
        i < (int)((nbytes-sizeof(phys_clicks))/(2*sizeof(phys_clicks))),
        temp != NIL_HOLE;
        temp = temp->h_next,
        i++) {

    sys_copy(MM_PROC_NR,
              D,
              (phys_bytes)&temp->h_len,
              mm_in.m_source,
              D,
              (phys_bytes)(buffer+2*i*sizeof(phys_clicks)),
              (phys_bytes)sizeof(phys_clicks)
             );

    sys_copy(MM_PROC_NR,
              D,
              (phys_bytes)&temp->h_base,
              mm_in.m_source,
              D,
              (phys_bytes)(buffer+(2*i+1)*sizeof(phys_clicks)),
              (phys_bytes)sizeof(phys_clicks)
             );
```

```
    }
    sys_copy(MM_PROC_NR,
             D,
             (phys_bytes)&a,
             mm_in.m_source,
             D,
             (phys_bytes)(buffer+(2*(i+1)*sizeof(phys_clicks))),
             (phys_bytes)sizeof(phys_clicks)
          );

    return (i);
 }
```

Drugie wywołanie systemowe `WORST_FIT` umożliwiające zmianę algorytmu alokacji. Jego funkcjonalność jest realizowana przez funkcję `do_worst_fit()`

```
/*===========================================================*
                      do_worst_fit
*===========================================================*/

PUBLIC int do_worst_fit(void)
{
     if((mm_in.m1_i1 == FIRST_FIT) ||
        (mm_in.m1_i1 == WORST_FIT))
         mem_alg = mm_in.m1_i1;

     return 0;
}
```

## 2.2. Oprogramowanie testujące

Testy rozwiązania zrealizowałem za pomocą przykładowych programów i skryptów zaprezentowanych na stronie prowadzącego przedmiot dr inż. Tomasza Jordana Kruka. Test opierał się na trzech programach `x` symulującego program realizujący obliczenia będące *de facto* okrojoną wersją polecenia `sleep`, `t` wyświetlającego liczbę i rozmiar bloków wolnych oraz `w` przyjmującego jako argument wywołania 1 albo 0 włączając lub wyłączając algorytm worst fit w systemie operacyjnym. Kod źródłowy tych tych programów został zamieszczony na stronie prowadzącego przedmiot.

Programy testowe zostały wykorzystane w skrypcie `lab` który pokazuje działanie mechanizmu alokacji w przypadku zastosowania obu algorytmów. Podstawowa wersja skryptu również znajduje się na stronie prowadzącego. W celu lepszej prezentacji działania algorytmu, dokonałem kilku prostych modyfikacji. Po pierwsze, oprócz wielkości dziury pamięci, prezentowany jest również adres początku dziury. Po drugie, dodałem wywołanie programu `t` wypisującą mapę pamięci również przed pierwszą iteracją pętli uruchamiającej programy `x`. Dzięki temu możemy stwierdzić czy ilość zalokowanej pamięci na zakończenie testu jest taka sama jak przed uruchomieniem pierwszego.

### 2.2.1. Algortym first fit

Algorytm first fit alokuje pamięć w pierwszym wolnym kawałku pamięci który ma wystarczający rozmiar. Dla potrzeb testów możemy założyć że implementacja algorytmu first fit jest poprawna, ponieważ została ona zrealizowana przez twórcę systemu i wyniki testu traktować jako referencyjne. W pierwszej

kolejności sprawdziłem czy ilość wolnej pamięci przed testem i po jest taka sama. Ilość wolnej pamięci przed testem wynosiła

$$68 + 17 + 21 + 90 + 128563 = 128759$$

clicków. Po zakończeniu testu ilość wolnej pamięci wynosiła

$$68 + 17 + 21 + 90 + 128563 = 128759$$

clicków. Ilość wolnej pamięci w obu momentach jest równa. Co więcej, wszystkie wolne segmenty mają takie same rozmiary i ich początki znajdują się w tych samych miejscach. Zgodnie z oczekiwaniami, liczba segmentów wolnej pamięci jest stała. Co każdą iterację, z najmniejszego wolnego segmentu większego niż 9 clicków, zabierane jest dokładnie 9 clicków pamięci na rzecz uruchamianego programu `x`. W drugiej pętli sprawdzamy mapy pamięci w momencie gdy programy dealokują swoją pamięć. Pojawia się nowa dziura, która powstała ponieważ elementy zaczynają dealokować swoją pamięć w kolejności w której ją zaalokowały.

### 2.2.2. Algorytm worst fit

Algorytm worst fit alokuje pamięć w największym znalezionym kawałku pamięci. W tym przypadku pamięc powinna być zawsze alokowana w ostatnim segmencie który jest największy. Zgodnie z oczekiwaniami, co iterację pętli testu, z największego segmentu jest pobierany kolejny kawałek na program. Między zaalokowaną pamięcią pojawiają się dodatkowe wolne segmenty wielkości 62 clicków. W algorytmie first fit ten efekt nie pojawiał się. Efekt ten jest konsekwencją sposobu uruchamiania procesów. W pierwszej kolejności jest alokowana pamięc dla kopii procesu testowego za pomocą polecenia `fork` a następnie program jest podmieniany przez polecenie z grupy `exec` alokując przy tym wymaganą ilość pamięci. W przypadku tego algorytmu alokacji, system operacyjny alokując pamięć dla nowego procesu, zawsze wybiera największy wolny segment i to z niego pobiera kawałek pamięci. W momencie gdy procesy dealokują pamięć, dziury łączą się ze sobą tworząc większe segmenty, dzięki czemu w ostatnim obiegu ilość dziur jest taka jak była przed pierwszą pętlą tego testu. Ilość wolnej pamięci przed testem wynosiła

$$68 + 17 + 21 + 90 + 128563 = 128759$$

clicków. Po zakończeniu testu algorytmu ilość wolnej pamięci wynosiła

$$68 + 17 + 21 + 90 + 128563 = 128759$$

clicków. Dziury które już były w pamięci przed testem pozostały niezmienione, ich wielkość i pierwszy adres pozostały takie same.

Pomiędzy dziurami tworzonymi w trakcie testu, odległości między początkami wynoszą 71 clicków. Każda z dziur ma 62 clicki szerokości. Wynika z tego że pomiędzy końcem jednej dziury a początkiem drugiej jest zaalokowanych dokładnie 9 clicków dla procesu `x`. Jest to dokładnie taka sama ilość alokowanej pamięci jak w przypadku referencyjnego algorytmu first fit. W drugiej pętli, pamięć procesów `x` jest dealokowana, co objawia się powiększaniem dziury pozostałej po pierwszej iteracji pierwszej pętli.

```
x: Stack+malloc area changed from 8000 to 8000 bytes.
-[ std ]------------------------------
[5]     68:112  17:186  21:345  90:497  128563:1357
[5]     68:112  6:197   21:345  90:497  128563:1357
[5]     68:112  6:197   12:354  90:497  128563:1357
[5]     68:112  6:197   3:363   90:497  128563:1357
[5]     68:112  6:197   3:363   81:506  128563:1357
[5]     68:112  6:197   3:363   72:515  128563:1357
[5]     68:112  6:197   3:363   63:524  128563:1357
[5]     68:112  6:197   3:363   54:533  128563:1357
[5]     68:112  6:197   3:363   45:542  128563:1357
[5]     68:112  6:197   3:363   36:551  128563:1357
[5]     68:112  6:197   3:363   27:560  128563:1357
[5]     68:112  15:188  3:363   27:560  128563:1357
[6]     68:112  15:188  9:345   3:363   27:560  128563:1357
[5]     68:112  15:188  21:345  27:560  128563:1357
[6]     68:112  15:188  21:345  9:497   27:560  128563:1357
[6]     68:112  15:188  21:345  18:497  27:560  128563:1357
[6]     68:112  15:188  21:345  27:497  27:560  128563:1357
[6]     68:112  15:188  21:345  36:497  27:560  128563:1357
[6]     68:112  15:188  21:345  45:497  27:560  128563:1357
[6]     68:112  15:188  21:345  54:497  27:560  128563:1357
[5]     68:112  17:186  21:345  90:497  128563:1357
-[ worst ]----------------------------
[6]     68:112  17:186  21:345  90:497  62:1222 128501:1419
[7]     68:112  17:186  21:345  90:497  62:1222 62:1295 128428:1492
[8]     68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 128357:1563
[9]     68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 128286:1634
[10]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 128215:1705
[11]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 128144:1776
[12]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 62:1650 128073:1847
[13]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 62:1650 62:1721 128002:1918
[14]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 62:1650 62:1721 62:1792 127931:1989
[15]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 62:1650 62:1721 62:1792 62:1863 127860:2060
[16]    68:112  17:186  21:345  90:497  62:1222 62:1295 62:1366 62:1437 62:1508 62:1579 62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[16]    68:112  17:186  21:345  90:497  62:1222 71:1286 62:1366 62:1437 62:1508 62:1579 62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[15]    68:112  17:186  21:345  90:497  62:1222 142:1286        62:1437 62:1508 62:1579 62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[14]    68:112  17:186  21:345  90:497  62:1222 213:1286        62:1508 62:1579 62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[13]    68:112  17:186  21:345  90:497  62:1222 284:1286        62:1579 62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[12]    68:112  17:186  21:345  90:497  62:1222 355:1286        62:1650 62:1721 62:1792 62:1863 62:1934 127789:2131
[11]    68:112  17:186  21:345  90:497  62:1222 426:1286        62:1721 62:1792 62:1863 62:1934 127789:2131
[10]    68:112  17:186  21:345  90:497  62:1222 497:1286        62:1792 62:1863 62:1934 127789:2131
[9]     68:112  17:186  21:345  90:497  62:1222 568:1286        62:1863 62:1934 127789:2131
[8]     68:112  17:186  21:345  90:497  62:1222 639:1286        62:1934 127789:2131
[6]     68:112  17:186  21:345  90:497  62:1222 128501:1419
-[ std ]------------------------------
```

Rysunek 2.1. Mapy pamięci dla algorytmów `FIRST_FIT` i `WORST_FIT`