Systemy Operacyjne

Zarządzanie Pamięcią

Kacper Górski

331379

Cel ćwiczenia

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.

Zadanie do zrealizowania

Zdefiniowanie dwóch dodatkowych funkcji systemowych, identyfikowanych stałymi *HOLE_MAP* oraz *WORST_FIT* w *include/minix/callnr.h*, zwiększenie stałej *NCALLS* o 2:

```
#define SVRCTL          77
#define HOLE_MAP        78
#define WORST_FIT       79
```

Funkcja systemowa *HOLE_MAP* umożliwiająca zdefiniowanie własnej funkcji

```
PUBLIC int hole_map( void *buffer, size_t nbytes)
{
    /* ... _syscall(..HOLE_MAP..) ... */
    message m;
    m.m1_p1 = buffer;
    m.m1_i1 = nbytes;
    return _syscall(MM, HOLE_MAP, &m);
}
```

która ma za zadanie zwrócić w buforze *buffer* o rozmiarze *nbytes* informacje o

aktualnej zawartości listy wolnych bloków utrzymywanej przez moduł zarządzania pamięcią (MM).

Funkcja systemowa *WORST_FIT* powinna umożliwiać wybór algorytmu wyboru elementu z listy wolnych bloków i zdefiniowanie własnej funkcji

```
PUBLIC int worst_fit( int w )
{
    /* ... _syscall(..WORST_FIT..) ... */
    message m;
    m.m1_i1 = w;
    return _syscall(MM, WORST_FIT, &m);
}
```

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.

Deklaracja prototypu w *src/mm/proto.h*

```
_PROTOTYPE( int do_hole_map, (void) );
_PROTOTYPE( int do_worst_fit, (void) );
```

Nowy wpis w *src/mm/table.c*

```
    do_hole_map,
    do_worst_fit,
};
```

Nowy wpis w *src/fs/table.c*

```
    no_sys,
    no_sys,
};
```

Funkcja *do_hole_map*

```
PUBLIC int do_hole_map()
/*
   The function should iterate over the list of memory holes (hole_head)
   and return a map of hole sizes and addresses
   The caller can then inspect the state of memory fragmentation
*/

{
    struct hole * hp;

    phys_clicks * buff = (phys_clicks *) mm_in.m1_p1;
    phys_clicks buffer[NR_HOLES * 2 + 1];

    size_t nbytes = mm_in.m1_i1;
    size_t max_pairs = nbytes / (2 * sizeof(unsigned int));
    unsigned int count = 0;

    for (hp = hole_head; hp != NIL_HOLE && count < max_pairs && hp->h_base < swap_base; hp = hp->h_next)
    {
        buffer[count * 2] = h_size;
        buffer[count * 2 + 1] = h_address;
        ++count;
    }

    if (count < max_pairs) {
        buffer[count * 2] = 0;
    }

    /* |SYS_COPY|src seg|src proc|src vir|dst seg|dst proc|dst vir|byte ct| */
    sys_copy(MM_PROC_NR, D,(phys_clicks) buffer, who, D, (phys_clicks) mm_in.m1_p1, (phys_clicks) mm_in.m1_i1);

    return count;
}
```

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 *do_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.

Funkcja *do_worst_fit*

Zdefiniowanie zmiennej globalnej *is_worst_fit* w pliku *src/mm/alloc.c*

```
PRIVATE int is_worst_fit = 0;

PUBLIC int do_worst_fit()
{
    int w = mm_in.m1_i1;
    if (w == 0)
        is_worst_fit = 0;
    else if (w == 1)
        is_worst_fit = 1;

        return 0;
}
```

Funkcja ma za zadanie zamianę algorytmu zarządzania pamięcią z worst fit na first fit, w zależności od tego jaka jest wartość zmiennej globalnej is_worst_fit.

Zmieniona funkcja *alloc_mem*

Funkcja ta jest odpowiedzialna za przydzielenie żądanej ilości pamięci (w jednostkach tzw. *clicks*, czyli minimalnych jednostek pamięci) z listy wolnych fragmentów.

Jeśli aktywowany jest algorytm worst_fit (is_worst_fit jest ustawione na jeden), funkcja alloc_mem wywołuje funkcję find_biggest_hole, aby znaleźć największy dostępny fragment, który pomieści żądaną ilość pamięci. Jeśli taki fragment zostanie znaleziony, jest on "przycinany" (jego początek zostaje przesunięty, a jego długość zmniejsza się o rozmiar przydzielonej pamięci). Jeśli fragment jest już w pełni wykorzystany, zostaje usunięty z listy wolnych fragmentów. Proces powtarza się do momentu, gdy uda się znaleźć odpowiedni fragment lub wyczerpie się dostępna pamięć. W przypadku braku wystarczającej ilości pamięci, funkcja próbuje "wymienić" inne procesy do przestrzeni swap (używając funkcji swap_out).

W przeciwnym wypadku (is_worst_fit jest ustawione na zero) aktywowany jest standardowy algorytm first fit.

Zmieniony kod funkcji *alloc_mem*

```
PUBLIC phys_clicks alloc_mem(clicks)
phys_clicks clicks;     /* amount of memory requested */
{
/* Allocate a block of memory from the free list using first fit. The block
 * consists of a sequence of contiguous bytes, whose length in clicks is
 * given by 'clicks'.  A pointer to the block is returned.  The block is
 * always on a click boundary.  This procedure is called when memory is
 * needed for FORK or EXEC.  Swap other processes out if needed.
 */

  register struct hole *hp, *prev_ptr, *prev_biggest_hole;
  struct hole * worst_hole = NIL_HOLE;
  phys_clicks old_base;
  if (!is_worst_fit) {
    do {
      hp = hole_head;
      while (hp != NIL_HOLE && hp->h_base < swap_base) {
          if (hp->h_len >= clicks) {
            /* We found a hole that is big enough.  Use it. */
            old_base = hp->h_base;  /* remember where it started */
            hp->h_base += clicks;   /* bite a piece off */
            hp->h_len -= clicks;    /* ditto */

            /* Delete the hole if used up completely. */
            if (hp->h_len == 0) del_slot(prev_ptr, hp);

            /* Return the start address of the acquired block. */
            return(old_base);
          }

          prev_ptr = hp;
          hp = hp->h_next;
      }
    } while (swap_out());       /* try to swap some other process out */
  }

  else {
    do {
        /* worst-fit allocation */
        worst_hole = find_biggest_hole(clicks, &prev_biggest_hole);

        if (worst_hole != NIL_HOLE) {
            /* use the biggest suitable hole */
            old_base = worst_hole->h_base;
            worst_hole->h_base += clicks;
            worst_hole->h_len -= clicks;

            /* remove hole if completely used up */
            if (worst_hole->h_len == 0) del_slot(prev_biggest_hole, worst_hole);

            return old_base;
        }
    } while(swap_out());
  }
  return(NO_MEM);
}
```

Funkcja pomocnicza znajdująca największą dziurę *find_biggest_hole*

Funkcja ta służy do znalezienia największego dostępnego fragmentu pamięci, który jest wystarczająco duży, by pomieścić żądany rozmiar pamięci. W pętli przechodzi przez listę wolnych fragmentów pamięci (reprezentowanych przez struktury hole), porównując długość każdego fragmentu z wymaganym rozmiarem. Jeśli fragment jest wystarczająco duży i większy niż dotychczas znaleziony największy, zostaje zapisany jako najlepszy kandydat. Zwraca wskaźnik do największego fragmentu, który spełnia wymagania, a także wskazuje na poprzedni fragment (jeśli taki istnieje), co jest przydatne do dalszego usuwania fragmentu z listy, gdy zostanie w pełni przydzielony

```
PRIVATE struct hole *find_biggest_hole(phys_clicks clicks, struct hole **prev_biggest_hole_ptr)
{
    struct hole *hp = hole_head;
    struct hole *biggest_hole = NIL_HOLE;
    struct hole *prev_ptr = NIL_HOLE;

    *prev_biggest_hole_ptr = NIL_HOLE;

    while (hp != NIL_HOLE && hp->h_base < swap_base) {
        if (hp->h_len >= clicks && (biggest_hole == NIL_HOLE || hp->h_len > biggest_hole->h_len)) {
            *prev_biggest_hole_ptr = prev_ptr;
            biggest_hole = hp;
        }
        prev_ptr = hp;
        hp = hp->h_next;
    }

    return biggest_hole;
}
```

Skrypt do testowania działania funkcji systemowych HOLE_MAP oraz WORST_FIT

```
#!/bin/sh
# skrypt do testowania działania funkcji systemowych
# HOLE_MAP oraz WORST_FIT
cc -o t t.c
cc -o w w.c
cc -o x x.c
chmem =8000 x

echo "-[ std ]---------------------------------------"
./w 0
for i in 1 2 3 4 5 6 7 8 9 10
do
    ./x 10 &
    ./t
    sleep 1
done
for i in 1 2 3 4 5 6 7 8 9 10
do
    ./t
    sleep 1
done
echo "-[ worst ]---------------------------------------"
./w 1
for i in 1 2 3 4 5 6 7 8 9 10
do
    ./x 10 &
    ./t
    sleep 1
done
for i in 1 2 3 4 5 6 7 8 9 10
do
    ./t
    sleep 1
done
echo "-[ std ]---------------------------------------"
./w 0
```

Pliki w.c, x.c oraz t.c

x.c – program służący do wywołania funkcji sleep

```c
#include <stdlib.h>
#include <unistd.h>

int
main( int argc, char *argv[] )
{
    if( argc < 2 )
        return 1;
    sleep( atoi( argv[1] ) );
    return 0;
}
```

t.c – program wyświetlający liczbę i rozmiary wolnych bloków

```c
/* t.c - polecenie t wyswietla liczbe i rozmiary blokow wolnych */
#include <stdio.h>
#include <unistd.h>
#include <lib.h>

PUBLIC int hole_map( void *buffer, size_t nbytes)
{
    /* ... _syscall(..HOLE_MAP..) ... */
    message m;
    m.m1_p1 = buffer;
    m.m1_i1 = nbytes;
    return _syscall(MM, HOLE_MAP, &m);
}

int
main( void )
{
        unsigned int    b[1024];
        unsigned int    *p, a, l;
        int     res;

        res = hole_map( b, sizeof( b ) );
        printf( "[%d]\t", res );
        p = b;
        while( *p )
        {
                l = *p++;
                a = *p++; /* tu niewykorzystywane */
                printf( "%d\t", l );
        }
        printf( "\n" );
        return 0;
}
```

w.c – program przełączający algorytmy zarządzania pamięcią

```c
/* w.c - polecenie w przyjmuje jako argument 1 albo 0 */
/* wlaczajac/wylaczajac algorytm worst fit w systemie Minix */
#include <stdlib.h>
#include <unistd.h>
#include <lib.h>


PUBLIC int worst_fit( int w )
{
    /* ... _syscall(..WORST_FIT..) ... */
    message m;
    m.m1_i1 = w;
    return _syscall(MM, WORST_FIT, &m);
}

int main( int argc, char *argv[] )
{
    if( argc < 2 )
        return 1;
    worst_fit( atoi( argv[1] ) );
    return 0;
}
```

Wynik działania skryptu

```
x: Stack+malloc area changed from 131072 to 8000 bytes.
-[ std ]--------------------------------------
[8] 5   6   38  46  62  28  62  128501
[7] 5   6   29  46  62  28  128563
[7] 5   6   20  46  62  28  128563
[7] 5   6   11  46  62  28  128563
[7] 5   6   2   46  62  28  128563
[7] 5   6   2   37  62  28  128563
[7] 5   6   2   28  62  28  128563
[7] 5   6   2   19  62  28  128563
[7] 5   6   2   10  62  28  128563
[7] 5   6   2   1   62  28  128563
[7] 5   15  2   1   62  28  128563
[8] 5   15  9   2   1   62  28  128563
[8] 5   15  18  2   1   62  28  128563
[8] 5   15  27  2   1   62  28  128563
[7] 5   15  38  1   62  28  128563
[8] 5   15  38  9   1   62  28  128563
[8] 5   15  38  18  1   62  28  128563
[8] 5   15  38  27  1   62  28  128563
[8] 5   15  38  36  1   62  28  128563
[7] 5   17  38  46  62  28  128563
-[ worst ]-------------------------------------
[9] 5   17  38  46  62  28  62  62  128428
[10]    5   17  38  46  62  28  62  62  62  128357
[11]    5   17  38  46  62  28  62  62  62  62  128286
[12]    5   17  38  46  62  28  62  62  62  62  62  128215
[13]    5   17  38  46  62  28  62  62  62  62  62  62  128144
[14]    5   17  38  46  62  28  62  62  62  62  62  62  62  128073
[15]    5   17  38  46  62  28  62  62  62  62  62  62  62  62  128002
[16]    5   17  38  46  62  28  62  62  62  62  62  62  62  62  62  127931
[17]    5   17  38  46  62  28  62  62  62  62  62  62  62  62  62  62  127860
[18]    5   17  38  46  62  28  62  62  62  62  62  62  62  62  62  62  62  127789
[18]    5   17  38  46  62  28  62  71  62  62  62  62  62  62  62  62  62  127789
[17]    5   17  38  46  62  28  62  142 62  62  62  62  62  62  62  62  62  127789
[16]    5   17  38  46  62  28  62  213 62  62  62  62  62  62  62  62  127789
[15]    5   17  38  46  62  28  62  284 62  62  62  62  62  62  127789
[14]    5   17  38  46  62  28  62  355 62  62  62  62  62  127789
[13]    5   17  38  46  62  28  62  426 62  62  62  62  127789
[12]    5   17  38  46  62  28  62  497 62  62  62  127789
[11]    5   17  38  46  62  28  62  568 62  62  127789
[10]    5   17  38  46  62  28  62  639 62  127789
[8] 5   17  38  46  62  28  62  128501
-[ std ]--------------------------------------
```

Algorytm first fit działa szybciej, ale powoduje większe rozdrobnienie pamięci, co utrudnia obsługę dużych procesów w dalszej części testów. Algorytm worst fit natomiast lepiej radzi sobie z utrzymaniem dużych bloków pamięci, co czyni go bardziej wydajnym w systemach wymagających obsługi dużych procesów.

Wartości końcowe sumy wolnej pamięci wskazują, że oba algorytmy przydzielają pamięć zgodnie z oczekiwaniami, jednak strategia przydziału znacząco wpływa na układ i dostępność wolnych bloków.

Wyniki skryptu pokazują różnice w efektywności i działaniu obu algorytmów. Worst fit jest bardziej efektywny w ograniczaniu fragmentacji i lepiej przygotowuje system na przyszłe, bardziej wymagające alokacje, podczas gdy first fit jest bardziej odpowiedni w sytuacjach wymagających szybkiego przydziału pamięci dla mniejszych procesów.