Project 5A: Memory Allocation Policies

Purpose:

The purpose of this project is to modify minix 3.1.2 memory system to implement several different allocation strategies to allocate space to processes competing for memory like the first fit, next fit, worst fit, best fit, and random fit. First fit allocation policy has already been implemented in minix and is currently in use. The Memory allocation describes how free space is managed.

- First fit chooses the first hole in which a segment will fit, starting its search from the beginning of the free list each time.
- Next fit is the same as first fit, except the search starts from the last hole you picked.
- Best fit chooses the hole that is the tightest fit.
- Worst fit chooses the hole that is loosest fit.
- Random fit chooses a random hole among those that the page will fit into, you can design any algorithm you would like to manage and select the random hole.

This project contains the following implementations:

1. A system call to select the allocation algorithm.
2. The various allocation algorithms.
3. A user process to collect and process statistics.

Along with this there are two programs used to generate the workload.

Resources modified:
System call: Made changes in the following files
/usr/src/servers/pm/ table.c - this file contains definitions for the call vec tablemapping the system call numbers onto the routines.
/usr/src/servers/pm/ proto.h - declare a prototype of the system call handler in file .Added the prototypes for the system call handlers to the proto.h.
/usr/src/servers/pm/ alloc.c - This file handles allocating and freeing memory. This file is used to modify (add other policies) the memory allocation policy for minix.
/usr/src/include/minix/callnr.h - map the system call number of the handler function to an identifier. The number assigned to the system call is 69 and 70.
/usr/include/minix/callnr.h - map the system call number of the handler function to an identifier.
Under the root directory added changeallocpolicy.c – this gives the user the option to choose from the different allocation policies.

Design

The data structure used is the linked list. The actual data structure is located in alloc.c. Hole array is of size NR HOLES that is set as 204. The first hole is given by hole head. The policies are explained below.
First fit policy :

```
// Iteration through the list of holes. Scan free list for first block
while (hp != NIL_HOLE && hp->h_base < swap_base) { //when the hole_head is not empty
if (hp->h_len >= clicks)  //checking for the length of the hole
{                                  //found a hole that is big enough to use
 old_base = hp->h_base;         // remember where it started
 hp->h_base += clicks; // bite a piece off
 hp->h_len -= clicks;    // ditto

 //for the used memory
 if(hp->h_base > high_watermark)
 high_watermark = hp->h_base;
 // Deleting the hole if it is used up completely
 if (hp->h_len == 0) del_slot(prev_ptr, hp);
 // Return the start address of the acquired block.
 last_hole = hp;
 return(old_base);
}
 prev_ptr = hp; //unlink
 hp = hp->h_next;
```

Next fit policy: similar to first fit except the search starts from the last hole. The pointer roves along the memory chain to search for a next fit. Thus each time a request is made the pointer begins searching from the place it last finished.

```
if(last_hole == NIL_HOLE) //checking form the last hole
{
  last_hole = hole_head; // if empty,set that as hole_head
  last_hole_prev = NIL_HOLE;
}
prev_ptr = last_hole_prev;
hp = last_hole;
```

Worst fit policy: This policy allocates the process to the largest available free block of memory. This leads to elimination of all large blocks of memory, thus requests of processes for large memory cannot be met.

```
if(hp->h_len >= clicks && hp->h_len > candidate->h_len) // finding the largest hole
```

Rest of the iteration would be the same as above.

```
{
  last_hole_prev = prev_ptr;
  candidate = hp;
}
prev_ptr = hp;
hp = hp->h_next;
last_hole_prev = NIL_HOLE;
```

Best fit policy: This policy allocates the process to the smallest available free block of memory. If large enough, then stop searching in the procedure if(hp->h_len >= clicks && hp->h_len < candidate->h_len) //opposite of next fit, equal or almost equal fit. Rest of the iteration would be the same as worst fit.

Random fit policy
First checks for the fitting holes and then randomly allocates holes. Rest of the iteration would be the same as the rest.

Top logic from top.c, the printmem function helped to get the statistics part of the project. Structure in type.h was also referred.

Testing

The allocation policy is chosen with the system call and the load is generated with the memuse and forkmem programs that use differing amounts of memory. The main program, memuse, will fork off a bunch of other processes that use memory in differing amounts for varying amounts of time. memlog.c – used to gather statistics regarding the number and the size of the holes. This information is gathered once per second and the number of holes, their average size, the standard deviation of their size, and the median of their size are computed. The memlog program takes name of a file to print the statistics as an argument. fopen and fprintf is used to print the lines to the log file. The value for time starts at 0 and increments each time a line is printed.