SOI: Laboratorium 5. - Zarządzanie pamięcią
Robert Wojtaś, 283234

Zadanie do zrealizowania

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.

Należy zaimplementować w systemie algorytm worst fit, a następnie zademonstrować i zinterpretować różnice w działaniu poszczególnych algorytmów.

Rozwiązanie

W ramach zadania należało utworzyć nowe wywołania systemowe:

- HOLE_MAP – funkcja zapisująca do bufora przekazanego jako argument pary (ROZMIAR, ADRES), będące informacjami na temat wolnych bloków pamięci.

- WORST_FIT – funkcja realizująca zmianę wartości pola worst_fit służącego do wyboru między standardowym algorytmem alokacji pamięci, algorytmem worst_fit.

Modyfikowane pliki:

1. /usr/include/minix/callnr.h

W pliku callnr.h została zmieniona liczba dostępnych wywołań systemowych oraz zdefiniowane zostały stałe HOLE_MAP I WORST_FIT.

2. /usr/src/mm/table.c

Dodane zostały nazwy funkcji obsługujących dane wywołania: do_hole_map oraz do_worst_fit.

3. /usr/src/mm/proto.h

Dodane zostały prototypy funkcji obsługujących dane wywołania:

- _PROTOTYPE(int do_hole_map, (void));

- _PROTOTYPE(int do_worst_fit, (void));

4. /usr/src/mm/alloc.c

Plik alloc.c jest odpowiedzialny za przydzielanie pamięci nowo powstałym procesom. Kluczową strukturą danych w pliku jest tablica tzw. dziur, która reprezentuje listę wolnych bloków pamięci.

W pliku zostało zdefiniowane I zainicjowane dodatkowe pole: PRIVATE int worst_fit = 0, które służy do wyboru między algorytmami alokacji pamięci.

W pliku zostały dodane funkcje obsługujące wywołania systemowe HOLE_MAP oraz WORST_FIT.

- int do_hole_map() - fukcja otrzymuje w wiadomości adres oraz rozmiar bufora, przesyłanego przez użytkownika. W funkcji tworzony jest tymczasowy bufor, do którego wpisywane są pary zawierające rozmiar I adres wolnych bloków pamięci. Ostatnią wartością wpisywaną do bufora jest 0 oznaczające koniec listy. Zawartość bufora tymczasowego jest kopiowana do bufora użytkownika za pomocą funkcji sys_copy. Funkcja zwraca liczbę par w buforze.

- int do worst_fit() - funkcja otrzymuje warość 1 lub 0 I w zależności od wartości przesłanej, ustawia wyżej wspomnianą wartość worst_fit na 1 lub 0 tym samym wybierając algorytm alokacji pamięci.

Kluczową zmianą w pliku była również modyfikacja funkcji alloc_mem(clicks), która na podstawie otrzymanej, żądanej ilości pamięci, przydziela ją procesowi za pomocą jednego z dwóch algorytmów: first_fit, worst_fit. Wybór zależy od wartości pola worst_fit.

Algorytm worst fit

Wybór największego możliwego bloku pamięci przebiega w pętli do{}while, która zawiera pętlę sprawdzającą wszystkie wolne bloki. Jeśli następna w kolejce dziura ma rozmiar większy od aktualnie sprawdzanej to jej adres przypisywany jest do wskaźnika max_hole będącego adresem największszej dziury. Następnie sprawdzana jest wielkość znalezionej dziury względem potrzebnej ilości pamięci I jeśli jest ona odpowiednia to pamięć jest przydzielana. Funkcja zwraca adres na zaalokowany blok pamięci.

Testy

W ramach testów zostały utworzone programy o zawartości podanej na stronie przedmiotu:

- w.c - polecenie w przyjmuje jako argument 1 albo 0, wywołuje funkcję systemową WORST_FIT przesyłając do niej podany argument

- t.c - polecenie t wyswietla liczbe i rozmiary blokow wolnych

- x.c - program pomocniczy x, okrojona wersja polecenia sleep

- **skrypt_testowy**

Skrypt został utworzony w celu testowania funkcji systemowych HOLE_MAP oraz WORST_FIT. Rozpoczyna się kompilacją wszystkich utworzonych wcześniej programów. Następnie występuje nowe polecenie: **chmem = 8000 x.**

Polecenie chmem w Minixie służy do zmian alokowanej pamięci. W tym przypadku powyższe wyrażenie oznacza przydzielenie procesowi **x** stosu o wielkości 8000 bajtów.

Kolejnym krokiem jest wybór standardowego algorytmu alokacji pamięci oraz wykonanie dwóch pętli for. Pierwsza z nich uruchamia 10 razy polecenia **x** oraz **t** po sobie w odstępach co sekundę. Następnie druga pętla for w ten sam sposób uruchamia jedynie polecenie **t.**

Gdy pętle zakończą działanie, wykonywane jest polecenie **w** zmieniające algorytm alokacji na zaimplementowany w ramach zadania worst fit. Działania po zmianie są analogiczne jak w przypadku piewszego algorytmu.

Dzięki wykonaniu polcenia **t,** możemy obserwować jakie zmiany zachodzą w każdym przebiegu każdej z pętli. Wynik programu tj. standardowe wyjście, zostało zapisane w pliku out.txt.

Różnice w działaniu algorytmów

x: Stack+malloc area changed from 131072 to 8000 bytes.
[5] 68 17 21 90 128563
-[std]----------
[5] 68 6 21 90 128563
[5] 68 6 12 90 128563
[5] 68 6 3 90 128563
[5] 68 6 3 81 128563
[5] 68 6 3 72 128563
[5] 68 6 3 63 128563
[5] 68 6 3 54 128563
[5] 68 6 3 45 128563
[5] 68 6 3 36 128563
[5] 68 6 3 27 128563
[5] 68 15 3 27 128563
[6] 68 15 9 3 27 128563
[5] 68 15 21 27 128563
[6] 68 15 21 9 27 128563
[6] 68 15 21 18 27 128563
[6] 68 15 21 27 27 128563
[6] 68 15 21 36 27 128563
[6] 68 15 21 45 27 128563
[6] 68 15 21 54 27 128563
[5] 68 17 21 90 128563
[5] 68 17 21 90 128563

[5] 68 17 21 90 128563
-[worst]----------
[7] 68 17 21 90 62 62 128428
[8] 68 17 21 90 62 62 62 128357
[9] 68 17 21 90 62 62 62 62 128286
[10] 68 17 21 90 62 62 62 62 62 128215
[11] 68 17 21 90 62 62 62 62 62 62 128144
[12] 68 17 21 90 62 62 62 62 62 62 62 128073
[13] 68 17 21 90 62 62 62 62 62 62 62 62 128002
[14] 68 17 21 90 62 62 62 62 62 62 62 62 62 127931
[15] 68 17 21 90 62 62 62 62 62 62 62 62 62 62 127860
[16] 68 17 21 90 62 62 62 62 62 62 62 62 62 62 62 127789
[16] 68 17 21 90 62 71 62 62 62 62 62 62 62 62 62 127789
[15] 68 17 21 90 62 142 62 62 62 62 62 62 62 62 127789
[14] 68 17 21 90 62 213 62 62 62 62 62 62 62 127789
[13] 68 17 21 90 62 284 62 62 62 62 62 62 127789
[12] 68 17 21 90 62 355 62 62 62 62 62 127789
[11] 68 17 21 90 62 426 62 62 62 62 127789
[10] 68 17 21 90 62 497 62 62 62 127789
[9] 68 17 21 90 62 568 62 62 127789
[8] 68 17 21 90 62 639 62 127789
[6] 68 17 21 90 62 128501
-[std]----------
[5] 68 17 21 90 128563

Powyższy wynik działania skryptu testowego został umieszczony w pliku tekstowym **out.txt.**

1. FIRST FIT

Za sprawą instrukcji **chmem** każdy proces x otrzymuje dla siebie przestrzeń 8000 bajtów, co według zaobserwowanych zmian wynosi około 9 klik. Na pierwszy rzut oka można stwierdzić, że algorytmy znacząco się od siebie różnią. W algorytmie first fit podczas wykonywania programu liczba dziur nie zwiększa się ponad 6. Co łatwo zauważyć rozmiary dziur wolnych zmniejszają się do momentu, w którym nie jest już możliwe użycie bloku dla procesu x. W przypadku wystąpienia takiej sytuacji pamięć dla procesu jest alokowana w następnym odpowiednim bloku.

Zastanawiające jest jednak zjawisko pozostawionej dziury o rozmiarze 68, która mogłaby być przecież wykorzystana jako potencjalne miejsce alokacji pamięci dla procesów **x**. Wytłumaczeniem tego problemu może być fakt, że to co obserwujemy na ekranie terminala jest stanem po uruchomieniu procesu **x**. To co w momencie wypisania na ekran jest wolną dziurą wcale nie musi nią być w trakcie tworzenia nowego procesu **x**. W Minixie pamięć alokowana jest na **fork** tworzący kopię procesu rodzica (w tym przypadku shell) oraz na **exec,** dla którego alokowana jest faktyczna pamięć potrzebna procesowi dziecku. W momencie tworzenia procesu **x** dziura o rozmiarze 68 jest zajęta przez proces shella dlatego pamięć na proces **x** wybierana jest z innego wolnego bloku o odpowiednim rozmiarze.

2. WORST FIT

W przypadku algorytmu **worst fit** możemy zaobserwować generowanie dziur o jednakowym rozmiarze w pierwszych 10 iteracjach. W każdej iteracji tworzony jest nowy proces trwający 10 sekund. Alokacja pamięci przebiega w jednakowy sposób (fork I exec) jednak zgodnie z algorytmem worst fit kolejne “kawałki” pamięci pochodzą z największej wolnej dziury. Możemy zauważyć, że przez pierwsze 10 iteracji ostatni podany rozmiar zmniejsza się o 71, co jest w przybliżeniu równe pamięci potrzebnej na utworzenie kopii procesu rodzica oraz procesu dziecka. Można stwierdzić, że wybór największego bloku przebiega poprawnie. Po wykonaniu **exec** pamięć zajmowana przez proces shella jest zwalniana, tworząc dziury o rozmiarze 62.

W następnych 10 iteracjach następuje kończenie się procesów pracujących w tle. Możemy zaobserwować zwalnianie poszczególnych bloków pamięci wraz z zakończeniem kolejnego procesu. Dziury przylegają do siebie dlatego przy ich zwalnianiu nasępuje połączenie dwóch bloków w jeden większy. Tym samym w trakcie ostatnich iteracji jedna z dziur się powiększa.

3. WYCIEK PAMIĘCI

Na poprzedniej stronie wytłuszczone linijki obrazują stan wolnych bloków pamięci przed I po rozpoczęciu testów konkretnego algorytmu. Jak widać stan został utrzymany przez wszystkie etapy pracy skryptu testowego, czyli nie odnotowane zostały żadne straty pamięci.