Warszawa, 06.01.2020

student: Patryk Jan Sozański

grupa: 2I5

SYSTEMY OPERACYJNE: LABORATORIUM NR 5

RAPORT Z WYKONANEGO ĆWICZENIA

**Treść zadania**

Celem ćwiczenia jest zmiana domyślnego algorytmu przydziału pamięci w systemie Minix. Należy umożliwić́ 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 zdefiniować́ dwie dodatkowe funkcje systemowe, identyfikowane stałymi HOLE\_MAP oraz WORST\_FIT.

Funkcja systemowa HOLE\_MAP powinna umożliwiać́ zdefiniowanie własnej funkcji o sygnaturze:

*int hole\_map(void ∗buffer , size\_t nbytes)*

która ma za zadanie zwrócić́ w buforze buffer o rozmiarze nbytes informacje o aktualnej zawartości listy wolnych bloków utrzymywanej przez moduł zarzadzania pamięcią̨ (MM). Struktura otrzymanej w buforze informacji powinna być́ następująca:

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

gdzie kolejne pary rozmiar-adres odpowiadają̨ 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 zwracać́ przesłaną liczbę̨ par rozmiar-adres. Należy zabezpieczyć́ się̨ przed przepełnieniem zadanego jako argument wywołania bufora i wypełnić́ go tylko liczbą par mieszczących się̨ w buforze dbając o zakończenie listy pozycją rozmiar = 0.

Funkcja systemowa WORST\_FIT powinna umożliwiać́ 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. Wartością̨ zwracaną powinno być́ zawsze 0.

**Sposób rozwiązania problemu**

Aby poprawnie zrealizować polecenie, należało zaimplementować algorytm WORST\_FIT, zapewnić użytkownikowi systemu możliwość zmiany algorytmu oraz udostępnić dwa dodatkowe wywołania systemowe.

W pierwszej kolejności w mm.h zdefiniowałem dwie stałe symbolizujące wybrany algorytm alokacji pamięci. W tym samym pliku zadeklarowałem zmienną, która przechowuje informacje o aktualnie wybranym sposobie alokacji pamięci.

Algorytm worst fit zaimplementowałem w pliku alloc.c w funkcji alloc\_mem. W tym samym pliku zdefiniowałem dwie funkcje realizujące funkcjonalności nowych wywołań systemowych. Wywołanie HOLE\_MAP realizuje funkcja do\_hole\_map(), a drugie wywołanie systemowe WORST\_FIT umożliwiające zmianę̨ algorytmu alokacji, jest realizowane przez funkcję do\_worst\_fit().

**Testowanie**

Testy rozwiązania zrealizowałem za pomocą̨ przykładowych programów i skryptów zaprezentowanych na stronie prowadzącego przedmiot dr inż. Tomasza Kruka. Test opierał się̨ na trzech programach: x - symulującego program realizujący obliczenia będące de facto okrojoną wersją polecenia sleep, t - wyświetlającego liczbę̨ 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 programów został zamieszczony na stronie prowadzącego przedmiot.

Programy testowe zostały wykorzystane w skrypcie skrypt, który pokazuje działanie mechanizmu alokacji w przypadku zastosowania obu algorytmów. Podstawowa wersja skryptu również̇ znajduje się̨ na stronie prowadzącego. W celu lepszej prezentacji działania algorytmu, dokonałem kilku prostych modyfikacji. Po pierwsze oprócz wielkości dziury pamięci, prezentowany jest również̇ adres początku dziury. Po drugie dodałem wywołanie programu t wypisującą mapę̨ pamięci również̇ 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.

1. **Algorytm 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, ponieważ̇ została ona zrealizowana przez twórcę̨ 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 w obu momentach była równa. Co więcej, wszystkie wolne segmenty miały takie same rozmiary i ich początki znajdowały się̨ w tych samych miejscach. Zgodnie z oczekiwaniami, liczba segmentów wolnej pamięci była stała. Co każdą iterację, z najmniejszego wolnego segmentu większego niż̇ 9 clicków, zabierane jest dokładnie 9 clicków pamięci na rzecz uruchamianego programu x. W drugiej pętli sprawdzane są mapy pamięci w momencie, gdy programy zwalniają swoją pamięć. Pojawia się̨ nowa dziura, która powstała, ponieważ̇ elementy zaczynają̨ zwalniać swoją pamięć w kolejności, w której ją zaalokowały.

1. **Algorytm worst fit**

Algorytm worst fit alokuje pamięć w największym znalezionym kawałku pamięci. W tym przypadku pamięć 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 pamięcią̨ pojawiają̨ się̨ 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 pamięć dla kopii procesu testowego za pomocą̨ 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 zwalniają pamięć, dziury łączą się̨ ze sobą̨, tworząc większe segmenty, dzięki czemu w ostatnim obiegu ilość dziur jest taka, jak była przed pierwszą pętlą tego testu. Dziury, które już̇ były w pamięci przed testem, pozostały niezmienione, ich wielkość i pierwszy adres pozostały takie same. Pomiędzy dziurami tworzonymi w trakcie testu, odległości między początkami wynoszą̨ 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 ilość́ alokowanej pamięci jak w przypadku referencyjnego algorytmu first fit. W drugiej pętli, pamięć procesów x jest zwalniana, co objawia się̨ powiększaniem dziury pozostałej po pierwszej iteracji pierwszej pętli.

**Wynik działania skryptu testującego**

