Systemy operacyjne - SOI

Laboratorium 2

Raport

Prowadzący: mgr. inż. Aleksander Pruszkowski

Autor: Jakub Sikora

Termin: wtorek 14:00 - 16:00

1. Treść zadania

Algorytm szeregowania dzielący procesy użytkownika na grupy: A, B. Zakłada się, że proces jest umieszczony w grupie: A - gdy jego identyfikator procesu jest podzielny bez reszty przez 2, B - gdy jego identyfikator procesu nie jest podzielny bez reszty przez 2. Należy dokonać niezbędnych modyfikacji funkcji systemowych umożliwiającą przenoszenie procesów pomiędzy powyższymi grupami. Proszę także wykonać usługę systemową o prototypie: int set_scheduler(int x); Która ustala proporcje czasowe w jakich scheduler ma wybierać do wykonania między zadaniami z grupy A i B. Argument x może przyjmować wartości 0…100, i oznacza on ile procent czasu dostanie zadanie A (zadanie B dostanie odpowiednio 100-x procent czasu). Usługa ma zwracać 0 gdy udało się wykonać zmianę proporcji czasowych, gdy zwróci -1, oznaczać będzie, że proces wywołujący tę usługę został przydzielony do klasy B i

procesom tej klasy nie wolno zmieniać proporcji. Opracować również klarowną i powtarzalną metodę weryfikacji poprawności rozwiązania (np.: za pomocą aplikacji testowych)

2. Planowane rozwiązanie

Każdy proces grupy `USER` będzie miał przypisaną grupę priorytetu `pri_group` określającą czy należy do grupy A czy do grupy B. Grupa priorytetu będzie ustalana w trakcie tworzenia procesu i będzie inicjalizowana na podstawie przypisanego identyfikatora procesu. W późniejszej fazie życia procesu, grupa priorytetu nie będzie w żaden sposób związana z numerem pid. Pozwoli to na przenoszenie procesów między powyższymi grupami. Zmiana grupy priorytetu będzie realizowana za pomocą nowego wywołania systemowego `void set_pri_group(void)`. Procesy będą szeregowane w ramach pojedynczej kolejki a ich długość wykonywania będzie się różniła w zależności od grupy. Czas wykonywania procesu będzie mógł zostać zmieniony za pomocą wywołania systemowego `int set_scheduler(int x)`. Będzie ona ustalać procent kwantu czasu jaki będzie przyznawany procesom grupy A (procesy grupy B będą otrzymywać odpowiednio 100 - procent_grupy_A procent czasu). Bazowy kwant czasu będzie stały.

3. Rzeczywiste rozwiązanie

Rzeczywista realizacja rozwiązania nie odbiega znacząco od planowanego. Zasadnicza różnica nastąpiła w miejscu przyznawania procesowi czasu procesora. W zaimplementowanym przeze mnie rozwiązaniu czas procesora jest odpowiednio wydłużany i przyznawany w funkcji `sched()`. Aby cała procedura mogła przyznawać zmienny kwant czasu procesora, dodałem do struktury procesu `proc` pole informujące ile jeszcze cykli powinno zostać przyznanych temu procesowi, zanim odda on sterowanie do następnego procesu z kolejki. W funkcji `sched()`, wartość ta jest dekrementowana a następnie sprawdzana. Jeśli osiagnie ona zero, to czas procesora otrzymuje kolejny proces a liczba przyznanych cykli zależy od zmiennej ustawianej za pomocą wywołania systemowego `set_scheduler`. Aby procesor nie został przyznawany na za długi okres czasu pojedynczemu procesowi, skróciłem stałą `MILISECONDS` znajdującą się w pliku `usr/src/kernel/clock.c`.

4. Realizacja rozwiązania

Modyfikacje funkcjonalne w plikach źródłowych modułu file system

- W pliku `table.c` dodałem odpowiednie pola `no_sys` odpowiadające dodanym przeze mnie wywołaniom systemowym (numery od 78 do 81).

Modyfikacje funkcjonalne w plikach źródłowych modułu memory manager

- W pliku `table.c` dodałem odpowiednie pola (kolejno `do_getprigroup`, `do_changeprigroup`, `set_scheduler`, `get_usagetime` odpowiadające dodanym przeze mnie wywołaniom systemowym (numery od 78 do 81).
- W pliku `proto.h` dodałem prototypy nowych wywołań systemowych
- W pliku `main.c` dodałem ciała wywołań systemowych `do_getprigroup`, `do_changeprigroup`, `set_scheduler` oraz `get_usagetime`. Wywołania te są interfejsem do wywołań mikrojądra, przekazują sterowanie, argumenty i pid oryginalnego wywołującego do kernela.

Modyfikacje funkcjonalne w plikach źródłowych modułu kernel

- W pliku nagłówkowym `proc.h` dodałem pola informujące o grupie priorytetu `pri_group` oraz drugie informujące o ilości

cykli które zostały jeszcze temu procesowi zanim zostanie mu odebrany procesor.

- W pliku `proc.c` zmieniłem postać funkcji `sched()` tak aby oddawała procesor następnemu procesowi tylko gdy zakończy się wymagana liczba cykli.
- W pliku `system.c` zamieściłem procedury wykonujące wywołania kernela. Odwołują się one do faktycznej tablicy procesów oraz ustawiają zmienne globalne kernela takie jak współczynnik podziału.

Modyfikacje dodatkowych plików nagłówkowych

- W pliku `com.h` dodałem informacje o nowych wywołaniach systemowych jądra systemu operacyjnego.
- W pliku callnr.h dodałem informacje o nowych wywołaniach systemowych modułu `MM`, pełniących rolę interfejsu do wywołań jądra.

5. Testowanie rozwiązania

Testowanie rozwiązania odbywa się za pomocą programu `soi2`. Uruchamia on 8 procesów za pomocą polecenia `fork()`. Każdy proces “symuluje” wykonywanie długich obliczeń poprzez dekrementację dużej zmiennej typu `unsigned long int`. Po zakończeniu wirtualnych obliczeń, proces zwraca ile czasu zajęła mu ta czynność. Testowanie algorytmu szeregowania odbywa się poprzez porównanie czasów wykonania próby kontrolnej (tylko jeden proces aktywny) oraz procesów testowych (osiem procesów aktywnych).

Dodatkowo, za pomocą polecenia `scheduler <arg>` można podać nowy stosunek podziału priorytetu grup A i B.