# Systemy Operacyjne – laboratorium

*Ćwiczenie 2 – Szeregowanie procesów*

## 1. Treść zadania

Celem zadania jest zaprojektowanie i zaimplementowanie mechanizmu szeregowania w systemie MINIX. Należy zmodyfikować standardową procedurę szeregującą zgodnie z założeniami opisanymi poniżej:

- proces posiada priorytet bazowy **BASE_PRI** i priorytet aktualny **ACT_PRI**;
- dwie zmienne systemowe **MAX_AGE** i **MIN_PRI** *(MAX_AGE > MIN_PRI)* dzielą szeregowane procesy na trzy kategorie:
    - **priorytet bazowy > MAX_AGE** - proces realizowany w reżimie pobłażania (z wyjątkiem procesów klas TASK i SERVER), priorytet bieżący zawsze równy bazowemu;
    - **priorytet bazowy > MIN_PRI, <= MAX_AGE** - proces realizowany w reżimie starzenia - proces któremu jest zabierany procesor otrzymuje priorytet bieżący równy bazowemu, wszystkie inne z tej grupy zwiększają priorytet bieżący o 1 (maksymalnie do MAX_AGE). Proces któremu jest zabierany procesor zostaje wstawiony do kolejki za innymi procesami o tym samym priorytecie bieżącym;
    - **priorytet bazowy < MIN_PRI** - proces realizowany w reżimie priorytetów statycznych z podziałem czasu – proces, któremu jest zabierany procesor jest wstawiany do kolejki za innymi procesami o tym samym priorytecie bieżącym, priorytet bieżący zawsze równy bazowemu.

## 2. Przyjęte założenia

- wartości początkowe zmiennych: **MAX_AGE = 1000, MIN_PRI = 100**;
- każdy nowy proces otrzymuje priorytety bazowy i bieżący równe **MIN_PRI**;
- system udostępnia nowe wywołanie - ustaw parametr szeregowania.

## 3. Proponowane rozwiązanie

W celu zaimplementowania rozwiązania spełniającego ww. warunki zadania, zmodyfikowałem w pierwszej kolejności poniższe pliki, aby zadeklarować potrzebne później zmienne oraz aby zapewnić ich poprawne wartości w utworzonym procesie po wywołaniu funkcji fork():

- **plik /usr/src/kernel/proc.h:**
    - w strukturze '**proc**' umieściłem zmienne przechowujące priorytet bazowy - **base_priority** oraz bieżący - **current_priority**;
    - zadeklarowałem zmienne systemowe **MAX_AGE** oraz **MIN_PRI**.
- **plik /usr/src/kernel/system.c:**
    - w funkcji **do_fork()** - wywoływanej w celu utworzenia nowego procesu - dodałem instrukcje inicjalizujące zmienne **base_priority** oraz **current_priority** wartością zmiennej systemowej **MIN_PRI**.

Następnie dodałem nowe wywołanie systemowe, które umożliwi modyfikację zmiennych systemowych **MAX_AGE, MIN_PRI** oraz priorytetu bieżącego procesu. Wywołanie powinno znajdować się w mikrojądrze, jednocześnie będąc wywoływane za pośrednictwem serwera MM lub FS. W pierwszej kolejności dodałem wywołanie dla serwera MM, modyfikując pliki:

- **plik /usr/include/minix/callnr.h:**
  - dodałem identyfikator nowego wywołania systemowego **SETPRI**;
  - zwiększyłem o jeden stałą **N_CALLS**.

- **plik /usr/src/mm/proto.h:**
  - umieściłem prototyp funkcji **do_setpri()**.

- **plik /usr/src/mm/table.c:**
  - na końcu tablicy **call_vec** wstawiłem adres (nazwę) funkcji **do_setpri()**.

- **plik /usr/src/fs/table.c:**
  - w analogicznym miejscu umieściłem adres pusty funkcji – **no_sys**.

- **plik /usr/src/mm/main.c:**
  - umieściłem właściwą treść wywołania **do_setpri()**.

Następnie dodałem wywołanie do mikrojądra wykonujące właściwe modyfikacje na wartościach zmiennych, poprzez modyfikację plików:

- **plik /usr/include/minix/com.h:**
  - w sekcji **SYSTASK** dodałem kod wywołania **SYS_SETPRI** – 22.

- **plik /usr/src/kernel/system.c:**
  - zdefiniowałem prototyp funkcji **do_setpri();**
  - w funkcji **sys_task()**, w instrukcji **switch(m.m_type)** dodałem wywołanie **SYS_SETPRI**;
  - umieściłem właściwą treść wywołania **do_setpri(message *m_ptr)**.

Po wykonaniu ww. czynności przygotowujących środowisko jądra, należało przejść do właściwej implementacji, tzn. modyfikacji funkcji **sched()**, oraz **ready()** znajdujących się w pliku **/usr/src/kernel/proc.h**:

- modyfikacja funkcji **shed()** polegała na zaimplementowaniu różnych akcji w zależności od wartości zmiennej **base_priority** w procesie znajdującym się na początku kolejki procesów, co odpowiada w zasadzie utworzeniu trzech oddzielnych kolejek dla procesów z różnych grup (koncepcja Three-Level-Scheduling):

  - proces posiadający **base_priority** większy od **AGE_MAX** pozostaje na początku kolejki, przy czym nie wywołujemy funkcji **pick_proc()**, a więc wspomiany proces nie zostaje 'zdjęty z procesora';

  - proces posiadający **base_priority** mniejszy niż **MIN_PRI** zostaje umieszczony w kolejce za procesami o tym samym lub większym priorytecie bieżącym, który jest zawsze równy priorytetowi bazowemu procesu;

  - proces posiadający **base_priority** w przedziale **[MAX_AGE, MIN_PRI)**, otrzymuje priorytet bieżący równy bazowemu, podczas gdy wszystkie inne procesy z tej grupy zwiększają swój priorytet bieżący o 1. Następnie proces zostaje umieszczony w odpowiednim miejscu w kolejce (analogicznie jak w powyższym przypadku).

- w przypadku funkcji **ready()**, zadanie sprowadzało się do modyfikacji algorytmu umieszczenia procesu ubiegającego się o czas procesora w odpowiednim miejscu kolejki (modyfikacji uległ tylko algorytm dla procesów klasy **USER**).

*Autor: Maciej Suchecki – grupa 3I4*