Ćwiczenie 1.

Kompilacja jądra i wywołania systemowe

# 1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z mechanizmem realizacji wywołań systemowych (ang. *system calls*, tłumaczone również jako funkcje systemowe).

W trakcie ćwiczenia należy dodać do systemu nowe wywołanie systemowe, zwracające numer (indeks) procesu w tablicy procesów. Do przeprowadzenia ćwiczenia niezbędna jest dodatkowa dyskietka.

# 2. Wiadomości ogólne

Wywołania systemowe są to funkcje realizowane przez jądro systemu operacyjnego w kontekście danego procesu. Jest to jedyny dopuszczalny dla procesów użytkowych sposób wejścia do jądra systemu. Przykładowymi wywołaniami systemowymi są: *READ, WRITE, FORK, EXIT*. Należy odróżnić wywołania systemowe od realizujących je funkcji bibliotecznych: *read(), write(), fork(), exit().*

# 3. Wywołania systemowe w systemie MINIX

W systemie MINIX wywołania systemowe są realizowane wewnątrz jednego z modułów-serwerów: *MM* lub *FS*, w zależności od rodzaju samego wywołania. Wszystkie wywołania dotyczące szeroko rozumianych operacji plikowych (np. *OPEN, READ, WRITE, IOCTL*) są obsługiwane przez *FS*. Pozostałe wywołania (np. *GETPID, FORK, EXIT*) są obsługiwane przez *MM*. W obu modułach, w plikach *table.c*, znajdują się tablice *call\_vec*, które określają, które wywołania systemowe są w danym module obsługiwane i jaka funkcja tym się zajmuje. Procesy użytkowe maja do swojej dyspozycji odpowiednie procedury biblioteczne (np. *open(), fork()* itd.).

# 4. Czynności wstępne

Zapoznaj się z plikami źródłowymi potrzebnymi do generowania jądra systemu Minix 2.0 (SM). Pliki znajdują się w następujących katalogach:

*/usr/src/kernel* - źródła wszystkich sterowników oraz kodu jądra

*/usr/src/mm* - źródła modułu Memory Menager (*MM*)

*/usr/src/fs* - źródła modułu File System (*FS*)

*/usr/src/tools* - źródła programów służących do łączenia części SM w jeden bootowalny program oraz do inicjowania systemu.

# 5. Dodanie do systemu nowego wywołania systemowego

Dodaj do systemu nowe wywołanie systemowe *GETPROCNR*, obsługiwane wewnątrz modułu *MM* i zwracające numer procesu w tablicy procesów dla procesu, którego identyfikator (*PID*) będzie argumentem tego wywołania.

W tym celu należy:

1. W pliku */usr/include/minix/callnr.h* dodać identyfikator nowego wywołania systemowego *GETPROCNR* i ewentualnie zwiększyć o jeden stałą *N\_CALLS*.
2. Napisać procedurę obsługi *do\_getprocnr* i umieścić ja na przykład w pliku */usr/src/mm/main.c*   
   Prototyp tej funkcji umieścić w pliku */usr/src/mm/proto.h*. Funkcja ta powinna porównać przekazany do niej identyfikator procesu z identyfikatorami procesów użytkowych znajdującymi się w tablicy *mproc*, plik */usr/src/mm/mproc.h*  
   Jeżeli identyfikatory te są takie same, to funkcja musi zwrócić numer (indeks) procesu w tablicy procesów, w przeciwnym razie należy zasygnalizować błąd *ENOENT* zdefiniowany w pliku */usr/include/errno.h*  
   W zdefiniowanej przez nas procedurze *do\_getprocnr* mamy możliwość odwoływania się do zmiennych typu *message*:

* *mm\_in* - zawiera dane przekazywane do wywołania,
* *mm\_out* - zawiera dane zwracane do procesu wywołującego.

W pliku */usr/src/mm/param.h* znajdują się definicje ułatwiające odwołania do elementów wyżej wymienionych struktur. Wartość zwracana przez naszą funkcję *do\_getprocnr* umieszczana jest w strukturze *mm\_out* automatycznie (w polu *m\_type*). Zapoznaj się ze znaczeniem pola *mp\_flags* struktury *mproc*, zwróć uwagę na flagę *IN\_USE*.

1. W pliku */usr/src/mm/table.c* w tablicy *call\_vec* w odpowiednim miejscu wstawić adres (nazwę) funkcji *do\_getprocnr*, zaś w pliku */usr/src/fs/table.c* w tym samym miejscu umieścić adres pusty funkcji, *no\_sys*.
2. Dokonać rekompilacji i przeładowania systemu Minix z nowym jądrem. Procedura ta ma różny przebieg w zależności od tego, czy ćwiczenie wykonywane jest w systemie Minix zainstalowanym na twardym dysku, w systemie Minix pracującym pod kontrolą emulatora Bochs, czy też wreszcie wykonywane jest w specjalnie przygotowanej dystrybucji systemu Minix w całości ładowanej do RAM-dysku.  
   Poniżej zostanie opisany proces kompilacji nowego jądra w ostatniej z wymienionych konfiguracji.  
   Rekompilacja i restart systemu Minix w środowisku ze specjalną wersją systemu Minix ładującą się w całości do RAM-dysku
   1. przejście do katalogu */usr/src/tools*
   2. d2. zapoznanie się z akceptowalnymi zleceniami dla programu *make* (czyli z zawartością pliku Makefile) oraz z zawartością skryptu *mkboot*,
   3. d3. kompilacja nowego jądra wraz z utworzeniem dyskietki startowej z nowym jądrem:  
      $ *make fdboot* (inne możliwości, m.in.: "*make hdboot*", sam "*make*" bez instalacji bloku ładującego)
   4. bardzo ważnym krokiem, który należy wykonać najpóźniej w tym momencie, jest zachowanie na zewnętrznym nośniku wszelkich zmian w źródłach, dokonanych w celu wykonania ćwiczenia laboratoryjnego.  
      Należy pamiętać, że po przeładowaniu systemu odtwarzana jest w pamięci RAM wersja systemu bez jakichkolwiek zmian wprowadzonych przez użytkownika.
   5. Po zachowaniu zmienionych wersji plików źródłowych należy wykonać restart systemu z wykorzystaniem utworzonego jądra.   
      *$ cd   
      $* *shutdown* (lub wciśnięcie <Ctrl-Alt-Del>)  
      *$ boot*
   6. Testowanie własności nowego jądra. Ewentualne załadowanie z dyskietki uprzednio zachowanych zmian źródeł systemu, a w szczególności zawartości plików nagłówkowych, które powinny być spójne dla jądra i przygotowywanych pod system z nowym jądrem aplikacji.

# 6. Program testujący

Napisz krótki program testujący poprawność działania zaimplementowanego wywołania systemowego. Zadaniem programu jest wywołanie zdefiniowanej funkcji systemowej dla zadanego jako argument wywołania programu testowego identyfikatora procesu oraz dla 10 kolejnych identyfikatorów liczbowych procesów. Przykładowo, wywołanie programu testowego z argumentem 100 powinno odpytać o pozycje w tablicy procesów dla procesów o identyfikatorach od 100 do 110. Dla każdego wywołania funkcji systemowej, program testujący powinien wyświetlać albo zwrócony indeks w tablicy procesów, albo też, w przypadku wystąpienia błędu, kod błędu ustawiany przez funkcję *\_syscall()* w zmiennej *errno*.

Aby wywołać funkcję systemową należy skorzystać z funkcji bibliotecznej *\_syscall (/usr/include/lib.h)*. Jej pierwszym argumentem jest numer serwera (*MM* lub *FS*), drugim numer wywołania systemowego, trzecim wskaźnik na strukturę *message*, w której umieszczamy dane dla wywołania.

Deklaracja struktury *message* znajduje się w *pliku/usr/include/minix/type.h*. Do przekazania numeru *PID* procesu zaleca się użycie pola *m1\_i1* tej struktury. Funkcję *\_syscall()* zdefiniowano w pliku */usr/src/lib/others/syscall.c*

W programie testującym wskazane jest zaimplementowanie funkcji o sygnaturze *int getprocnr( int )* wewnętrznie wykorzystującej *\_syscall()* do uzyskania pozycji procesu w tablicy procesów. Poprawność rezultatu zwracanego przez wywołanie zdefiniowanej funkcji systemowej można weryfikować analizując aktualną zawartość tablicy procesów oraz wykorzystując komendę ps. Tablicę procesów można wyświetlić na pierwszej konsoli poprzez naciśnięcie klawisza F1. Analogicznie poprzez wciśnięcie F2 uzyskuje się aktualną mapę pamięci.