1 Zapoznanie się ze środowiskiem Xenomai.

Wszystkie ćwiczenia oraz programy opracowane zostały w Xenomai w wersji 2.5.6. Dlatego też odwołania do dokumentacji dotyczą dokumentu pod adresem:
`http://www.xenomai.org/documentation/xenomai-2.5/html/api/`
Jest to oficjalna dokumentacja projektu, która stanowi podstawowy materiał do nauki. Dodatkowo używany będzie tylko interfejs natywny (Native Xenomai API). Aby uzyskać do niego dostęp należy po wejściu na stronę dokumentacji wybrać kolejno **Modules/** i **Native Xenomai API/** z menu znajdującego się po lewej stronie. Inne źródła informacji na temat Xenomai podane są poniżej:
`http://www.xenomai.org/documentation/xenomai-2.5/pdf/native-api.pdf`
`http://git.xenomai.org/?p=xenomai-2.5.git;a=tree;f=include/native`
Dokumentacja dostępna jest również lokalnie w katalogu:
`/usr/xenomai/share/doc/xenomai/`
Źródła (pliki nagłówkowe) znajdują się natomiast w katalogu:
`/usr/xenomai/include/native/`

W celu sprawdzenia poprawności działania systemu i jego komponentów Xenomai dostarcza zbiór kilku programów konfiguracyjnych oraz testowych. Aby upewnić się co do wersji zainstalowanego systemu należy wykonać polecenie:

```
$ xeno-config --version
2.5.6
```

Program ten najbardziej jest jednak przydatny na etapie kompilacji własnych programów. „Pamięta" prawidłowe ścieżki do bibliotek, plików nagłówkowych, oraz niezbędne flagi. Dodatkowo przechowuje również nazwę zalecanego kompilatora. Środowisko Xenomai oferuje szereg interfejsów programistycznych, z których jeden należy wybrać i wskazać w czasie kompilacji. Wszystkie prezentowane tutaj ćwiczenia i programy zostały wykonane przy użyciu interfejsu „native". Flagi `CFLAGS` i `LDFLAGS` trzeba sparametryzować nazwą interfejsu, którego chcemy użyć. Polecenie wypisujące flagi dla kompilatora i linkera:

```
$ /usr/xenomai/bin/xeno-config --skin=native --cflags
-I/usr/xenomai/include -D_GNU_SOURCE -D_REENTRANT -Wall -pipe -D__XENO__
$ /usr/xenomai/bin/xeno-config --skin=native --ldflags
-lnative -L/usr/xenomai/lib -lxenomai -lpthread
```

Informacje o systemie, jego zasobach, wątkach oraz wszystkich obiektach utworzonych przez Xenomai znajdują się w postaci plików i katalogów w systemie plików /proc, a dokładnie w `/proc/xenomai/`. Pliki `sched i stat` zawierają informacje o uruchomionych procesach RT. Wpisując polecenie `cat sched` można zauważyć, że uruchomiony jest jeden proces o nazwie `ROOT/0` o priorytecie -1. Jest to Linux. Wydając polecenie `cat stat` wyświetlone zostaną dodatkowe informacje o nim, takie jak zużycie procesora. Uruchamiając teraz jeden z programów testowych Xenomai można zauważyć zmiany w plikach w /proc/xenomai/ spowodowane aktywnością systemu. Program `switchtest` służy do testowania zmian kontekstu wykonywanych zadań. W pierwszym terminalu należy wpisać:

```
$ /usr/xenomai/bin/switchtest -s50 rtup rtup
```

Po wydaniu tego polecenia naturalne będzie zauważalne spowolnienie reakcji systemu. Spowodowane jest to zamierzonym dodatkowym obciążeniem dodanym parametrem -s50. W drugim terminalu można sprawdzić zawartość plików `sched i stat`. Jak widać dodane zostały 3 nowe wątki, a proces ROOT/0 nie stanowi już 99czasu procesora.

Polecenie `$ dmesg` wyświetli informacje o wszystkich utworzonych i usuniętych obiektach. W przypadku błędów pojawią się tam również ich kody. Natomiast kody błędów zwracanych przez funkcje natywnego API można rozkodować na podstawie plików:

```
/usr/include/asm-generic/errno-base.h
/usr/include/asm-generic/errno.h
```

Kompilacja i uruchomienie programu. Zapoznaj się z plikiem `src/ex1/ex1_main.c`. W pliku znajdują się komentarze opisujące wszystkie wykonywane instrukcje. Następnie skompiluj go i uruchom według podanych niżej poleceń:

```
$ export XCFG=/usr/xenomai/bin/xeno-config
$ export CFLAGS=‘$XCFG --skin=native --cflags‘
$ export LDFLAGS=‘$XCFG --skin=native --ldflags‘
$ gcc $CFLAGS $LDFLAGS -lrtdk ex0_main.c -o main
$ ./main
```

Program zawiera umyślnie umieszczoną instrukcję, która powoduje niepożądane opóźnienia - printf. Spowodowane to jest znajdującymi się w niej wywołaniami systemowymi, które prowadzą do przełączenia trybu wykonywania. Ten skok oznacza, że Xenomai nadaje Linuksowi najwyższy

priorytet i wykonuje jego procesy, po czym wraca do normalnego trybu. Program po uruchomieniu wyświetla 3 informacje: nazwę zadania, liczbę przełączeń trybu, oraz czasy wykonywania zadania. Jak widać licznik przełączeń wskazuje na 1 z powodu znajdującego się printf. Uruchom program kilka razy i zwróć uwagę na czas wykonywania. Następnie usuń (zakomentuj) printf i uruchom program ponownie kilka razy. Można zauważyć, że czasy wykonywania wyraźnie się zmniejszyły.

Korzystając z dokumentacji (źródła podane na początku) sprawdź jakie jeszcze informacje można odczytać z `RT_TASK_INFO` oraz wypisz je (`Task management services / rt_task_inquire`).

2 Uruchamianie zadań i dostęp do timera systemowego.

Odpowiednikiem procesu w systemie czasu rzeczywistego jest zadanie. Posiada własny stos, jest odpowiedzialne za wykonanie fragmentu kodu w czasie przydzielanym mu przez planistę (scheduler) RT. Xenomai jest systemem wielozadaniowym z szeregowaniem zadań opartym na priorytetach.

Tworzenie oraz startowanie zadania. Poniżej znajduje się deklaracja funkcji odpowiedzialnej za utworzenie zadania.

```
int rt_task_create
(RT_TASK *task, const char *name, int stksize, int prio, int mode);
```

Pierwszy argument `task` przyjmuje adres wcześniej zadeklarowanego obiektu typu `RT_TASK`, w którym przechowywane są informacje o zadaniu niezbędne podczas jego wykonywania. Każde zadanie musi posiadać unikalną nazwę, którą przekazuje się parametrem `name`. Służy ona do identyfikacji zadania w specjalnej przestrzeni nazw środowiska Xenomai. Pozostałe argumenty określają rozmiar stosu (w bajtach), priorytet (0-99) oraz flagi ustawień. Szczegółowy opis tych parametrów znajduje się w dokumentacji w opisie omawianej funkcji (`Task Management Services` / `rt_task_create`). Wszystkim domyślnie można przypisać wartość 0. Wartość 0 dla stosu oznacza automatyczny przydział potrzebnej liczby bajtów przez Xenomai.

Gdy zadanie zostało utworzone i zarejestrowane w globalnej przestrzeni nazw znajduje się w stanie gotowości. Przejście do stanu wykonywania zadania następuje dzięki wywołaniu niżej podanej funkcji.

```
int rt_task_start
(RT_TASK *task, void(*entry)(void *cookie), void *cookie)
```

Pierwszym parametrem jest identyfikator utworzonego wcześniej zadania. Drugi parametr określa adres funkcji, która będzie punktem startowym zadania, czyli rzeczywiście wykonywanym fragmentem kodu w jego obrębie. Trzeci argument służy do przekazania dowolnej paczki danych do zadania. Funkcja obsługi zadania musi przyjmować odpowiedni wzorzec:

```
void func(void *arg)
{
  /* routine */
```

```
}
```

Kompilacja i uruchomienie programu. Zapoznaj się z zawartością plików ex2_main.c i Makefile znajdujących się w katalogu `src/ex2/`. Plik ex2_main.c zawiera najprostszy przykład użycia wcześniej opisanych funkcji. Tworzy oraz startuje zadanie czasu rzeczywistego, które następnie w cyklu 1s wypisuje na ekran rzeczywisty, zmierzony czas między kolejnymi wywołaniami. Za pomocą poniższych poleceń skompiluj oraz uruchom program. Aby przerwać jego działanie wciśnij `^C`.

```
$ make
$ ./ex2_main
```

(a) Za pomocą zmiany `SLEEP_TIME` sprawdź poprawność wypisywanego czasu dla kilku mniejszych wartości (np. 100ms, 10ms, 1ms). Przy czym należy pamiętać, żeby nie schodzić zbyt nisko, wartości poniżej 100us mogą być już traktowane jako działanie w pętli nieskończonej i przy obecnej wersji systemu powodować jego zawieszenie.

Kluczowym składnikiem każdego zadania jest jego priorytet. Określa on wagę wykonywanych przez nie operacji, a tym samym pierwszeństwo w dostępnie do najważniejszego zasobu procesora, czyli czasu.

(b) Dodaj do poprzedniego programu nowe zadanie (task2) w analogiczny sposób jak zostało stworzone zadanie task. Nie zapomnij o zmianie nazwy na nową, oraz o funkcji usunięcia zadania (`rt_task_delete`). W dokumentacji systemu zapoznaj się z funkcjami `rt_task_sleep` (Task management services), oraz `rt_timer_spin` (Timer management services). Stwórz funkcję `func2` i przypisz ją jako punkt startowy zadania `task2`.

```
void func2(void *arg)
{
  while(1) {
    time2 = rt_timer_read();
    rt_timer_spin(SLEEP_TIME);
    rt_task_sleep(SLEEP_TIME);
    rt_printf("2 %12lld\n\n", (rt_timer_read()-time2));
  }
}
```

`SLEEP_TIME` ustaw na 1s, a okres zadania `task1` na `SLEEP_TIME/2`. Skompiluj i uruchom program. Na ekranie rozkład zadań powinien wyglądać tak jak przedstawiono poniżej.

```
1    1000171639
1     498298511
1     499962431
2    2000039025

1    1000198777
1     499858787
1     499796832
2    2000102671
...
```

Pytanie sprawdzające. Wyjaśnij dlaczego pierwsze zadanie (task1) wyświetla kolejne czasy swojego wywołania w przedstawiony sposób. Dlaczego odstępy czasowe pomiędzy tymi wywołaniami różnią się?