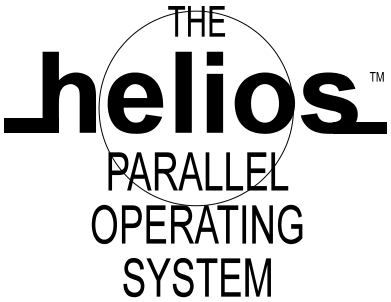
Politechnika Śląska
Instytut Informatyki
Zakład Oprogramowania

Równoległy system operacyjny Helios

Laboratorium programowania współbieżnego



Opracowali
Piotr Fabian i Wojciech Mikanik

Marzec 1996

Spis treści

1 Wstęp **3**

2 Charakterystyka systemu **4**
2.1 Nazywanie obiektów systemu 4
2.2 Serwer `I/O` 5
2.3 Nucleus 7

3 Interakcja z systemem - podstawy **9**
3.1 Uruchamianie systemu 9
3.2 Podstawowe operacje 10

4 Tworzenie oprogramowania w systemie Helios **12**
4.0.1 Program `make` 12
4.1 Edytor tekstowy MicroEmacs 14

5 Programowanie równoległe w systemie Helios **16**
5.1 Model obliczeń 16
5.2 Zadania 16
5.3 Komunikacja między zadaniami w systemie Helios 16
5.4 Język CDL 17
5.4.1 Deklaracja własności zadań 18
5.4.2 Deklaracja połaczeń między zadaniami 19
5.4.3 Replikatory 19
5.5 Kompilator języka CDL 20

6 Zadania **20**
Literatura

1 Wstęp

Helios jest wieloużytkownikowym, wielozadaniowym systemem operacyjnym zaprojektowanym z myślą o komputerach równoległych typu MIMD z rozproszoną pamięcią. System ten jest dostępny dla kilku różnych platform sprzętowych i kilku różnych procesorów, choć stosowany jest głównie(?) w instalacjach transputerowych. Twórcy systemu zapewnili jego częściową zgodność ze standardem Posix[1]. Na pierwszy rzut oka przypomina Unixa.

Procesory w sieci mogą pracować w dwóch trybach: jako zarządzane przez system Helios oraz w trybie „surowym" (ang. raw). W każdym procesorze zarządzanym przez Helios znajduje się kopia tzw. Nucleusa, zawierającego jądro systemu (ang. kernel), podstawowe biblioteki, Process Manager oraz program ładujący (ang. loader). Składowe Nucleusa omówiono w punkcie 2.3. Dodatkowe serwery (programy usługowe patrz 2.1) np: serwer systemu okien graficznych XWindows, serwer protokolu TCP/IP itd. mogą być wykonane w dowolnym procesorze sieci (o ile dysponuje on odpowiednim osprzętem, np: monitorem graficznym w przypadku serwera XWindows).

Niniejsza instrukcja zawiera podstawowe informacje pomocne przy wykonywaniu ćwiczeń w ramach laboratorium programowania współbieżnego. Zakłada się, że teoretyczny aspekt omawianych zagadnień jest Czytelnikowi doskonale znany.

Instrukcja składa się z sześciu części. W punkcie drugim przedstawiono ogólną charakterystykę systemu Helios[2] w wersji 1.3. Punkt trzeci opisuje podstawy obsługi systemu. Punkt czwarty zawiera informacje pomocne przy tworzeniu i wykonywaniu programów w tym systemie.Punkt piąty dostarcza informacji niezbędnych dla tworzenia programów równoległych w systemie Helios. W ostatnim punkcie podano treści zadań, które należy wykonać w trakcie ćwiczeń laboratoryjnych.

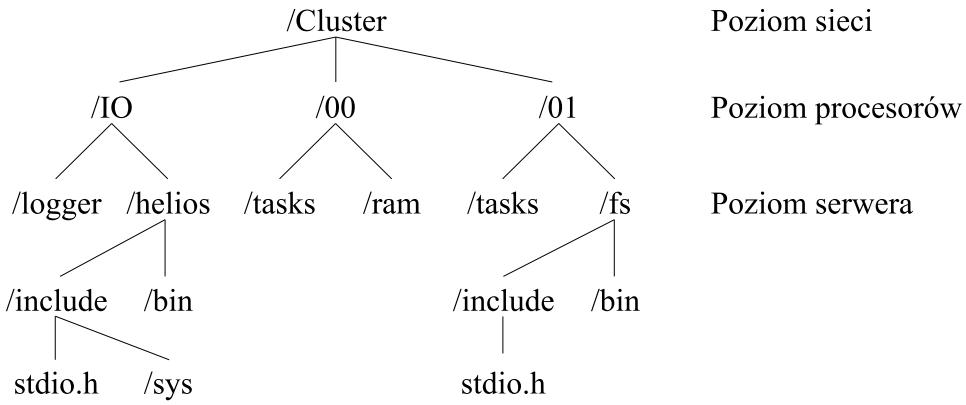
[1] Standard IEEE 1003.1-1988

[2] Helios jest znakiem towarowym fimry Perihelion Software Ltd.

2 Charakterystyka systemu

2.1 Nazywanie obiektów systemu

Przed opisaniem poszczególnych elementów Heliosa konieczne jest zapoznanie się z przyjętym schematem nazywania obiektów systemu. Zilustrowany jest on na rysunku 1.



Rysunek 1: Schemat nazywania obiektów systemu Helios

Na szczycie drzewa znajduje się jeden poziom (lub więcej) nazw **sieci**. W przykładzie sieć `/Cluster` stanowi korzeń schematu nazw. Poniżej poziomu sieci (lub poziomów) znajduje się poziom **procesorów**. Każdemu procesorowi nadawana jest nazwa podczas startu systemu. Nazwami zwykle są numery, ale nic nie stoi na przeszkodzie, aby nazywać procesory takimi nazwami jak `tom, dick, harry` itp. Poniżej poziomu procesorów znajduje się poziom **serwera**. Na przykład w procesorze `00` działają dwa serwery: `tasks` i `ram`, może ich być też więcej. Serwer udostępnia pewnego rodzaju usługi. Na przykład serwer plików wykonuje niektóre operacje wejścia/wyjścia. Zwykle każdy serwer jest oddzielnym programem, chociaż możliwa jest sytuacja, gdy pojedynczy program służy jako kilka serwerów.

Poniżej poziomu serwera znajduje się poziom zwykłego katalogu. Na przykład serwer nazwany `/Cluster/IO/helios` zawiera zwykłe katalogi `include` i `bin`, a katalog `include` zawiera podkatalog `sys` i plik `stdio.h`. Wszystkie poziomy przedstawionej hierarchii nazw obsługiwane są w ten sam sposób. Rozważmy polecenie `ls`, używane do wypisywania zawartości katalogu. Pole-

cenie `ls /Cluster/IO/helios/include` wypisze zwykłą zawartość katalogu (jak można się spodziewać). Polecenie `ls /Cluster/00` wypisze spis serwerów działających na procesorze `00`[3].

Na rys. 1 są przedstawione dwa pliki `stdio.h` przechowywane w dwóch oddzielnych serwerach plików. Mają różne adresy sieciowe: pierwszy z nich `/Cluster/IO/helios/include/stdio.h`, drugi `/Cluster/02/fs/include/stdio.h`. Wszystkie obiekty w przedstawionym schemacie nazywania mają unikalne nazwy sieciowe. Użytkownik sam musi zadbać o zgodność treści obu plików tak, by kompilator C mógł wykorzystać dowolny z nich. Najprawdopodobniej kompilator użyje `/01/fs`, aby uniknąć przesyłania danych przez łącze procesora `I/O`.

Specyfikacja obiektu niekoniecznie jest jego pełną nazwą sieciową. Zawsze musi być podana co najmniej nazwa serwera oraz pełna ścieżka w obrębie tego serwera.

Przykładowo, `/helios/include/stdio.h` jest poprawną nazwą Nie jest nią `/stdio.h`, ponieważ nie zawiera wystarczającej informacji o lokalizacji pliku. Skracanie nazw może wprowadzić wieloznaczność podanej nazwy. W sieci mogą na przykład znaleźć się dwa serwery nazwane `/helios`: jeden w procesorze `IO`, drugi w procesorze `01` z serwerem plików zainstalowanym jako `/helios` zamiast `/fs`. W takim przypadku `/helios/include/stdio.h` zwykle dotyczyć będzie najbliższego serwera, co niekoniecznie będzie poprawną sytuacją. Dwuznaczność może zawsze być usunięta przez podanie pełniejszej (lub pełnej) nazwy sieciowej obiektu.

Użytkownik może stworzyć własne serwery pisząc programy stosujące się do protokołu używanego między klientem a serwerem (general server protocol). Nie zawsze jest to łatwe, dlatego wielu użytkowników Heliosa zadowala się używaniem standardowych serwerów.

2.2 Serwer `I/O`

Serwer `I/O` jest główną i często jedyną częścią Heliosa działającą w procesorze macierzystym (host), a nie w węźle sieci (transputerze). Serwer `I/O` spełnia dwa główne zadania: inicjalizacja pierwszego procesora sieci i dostarczanie

[3]Ściśle mówiąc nie jest to prawda. Ze względu na efektywność każdy procesor przechowuje tablicę znanych nazw serwerów i procesorów, a polecenie `ls` wypisuje odpowiednie pozycje z tej tablicy. Dlatego pewne nazwy nie ukażą się w spisie, ponieważ nie było jeszcze do nich odwołań i nie sŃ one znane.

usług umożliwiających dostęp do zasobów procesora `I/O` (klawiatury, dysku itp.).

Serwer `I/O` jest programem dużym, ale łatwo konfigurowalnym. Plikiem konfiguracyjnym czytanym podczas staru systemu jest `host.con`. Zawiera on opis używanego sprzętu (sieci), lokalizację plików systemowych Heliosa i informację, czy ma być obsługiwana myszka.

Każdy procesor sieci oraz procesor `I/O` musi otrzymać nazwę podczas startu. Nazwy mogą pochodzić z pliku `host.con`, chociaż w przypadku braku odpowiednich wpisów zostaną użyte nazwy `/00` oraz `/IO`. Pierwszym zadaniem serwera `I/O` jest startowanie pierwszego procesora w sieci. Wykonanie tego zadania składa się z kilku etapów:

1. zainicjowanie sprzętu;
2. wyzerowanie (reset) *root processora*, co może mięć efekt uboczny w postaci wyzerowania części lub całej sieci transputerów;
3. wysłanie krótkiego programu ładującego `nboot.i`. Wykonuje on inicjalizację sprzętu od strony transputera i czeka na dalsze polecenia serwera `I/O`;
4. wysłanie do programu ładującego Nucleusa. Program ładujący przekazuje sterowanie pierwszemu elementowi Nucleusa, Kernelowi;
5. wysłanie dodatkowych informacji konfiguracyjnych wymaganych przez Kernela.

Po wykonaniu tych czynności Helios działa w *root processorze*, a procesor `I/O` staje się biernym elementem systemu. Działają w nim serwery udostępniające różnego rodzaju operacje wejścia/wyjścia dla sieci transputerów. Jakie to będą serwery, zależy od komputera macierzystego (hosta). Dla typowego PC najczęściej są to:

- `/logger` - zapisuje komunikaty błędów;
- `/window` - system pseudo-okienkowy pozwalający przełączać sesje;
- `/helios` - serwer plików udostępniający pliki systemu Helios;
- `/a`, `/c`, `/d` - dodatkowe serwery plików dla stacji dysków a:, c: i d:;

- `/rs232` - udostępnia porty szeregowe;
- `/centronics` - udostępnia porty równoległe;
- `/pc` - komunikacja między programami uruchamianymi pod kontrolą Heliosa a programami na PC.

Dla nadrzędnego komputera z systemem Unix:

- `/logger` - j.w.;
- `/window` - system okienek, używający terminala X Windows;
- `/helios` - j.w.;
- `/file` - serwer plików z Unix-owym systemem plików;

Wszystkie wymienione serwery są częścią sieci Heliosa. Nie jest konieczne posiadanie w sieci procesora `I/O`. Alternatywą jest system startowany z ROMu, skąd przesyłany jest do jednego z procesorów sieci program ładujący. Zwykle następuje potem ładowanie i uruchomienie Nucleusa z pierwszej ścieżki dysku twardego.

2.3 Nucleus

Nucleus jest częścią systemu obecną w każdym procesorze sieci. Serwer `I/O` ładuje Nucleusa do *root processora*. Później oprogramowanie sieciowe ładuje Nucleusa do każdego z pozostałych procesorów sieci. Nucleus składa się sześciu części: Kernela, biblioteki systemowej, bilblioteki serwera, biblioteki narzędziowej, zarządcy procesów oraz loadera. Zależności pomiędzy tymi częściami oraz między nimi a resztą systemu Helios przedstawiono na rys. 2.

- **Kernel** jest odpowiedzialny za bezpośrednią obsługę sprzętu. W przypadku transputera jest to obsługa komunikacji i monitorowanie zdarzeń. Dla każdego łącza prowadzącego do innego procesora z działającym systemem Helios uruchamiany jest proces oczekujący na komunikaty z tego procesora i przesyłający je do celu (np. innego łącza). Dodatkowo podczas startu Kernel wykrywa ilość pamięci zainstalowanej w transputerze i inicjalizuje system alokacji pamięci. Kernel obsługuje niskopoziomowe operacje takie jak przesyłanie komunikatów,

synchronizacja semaforów oraz tworzenie i usuwanie procesów. Zwykłe aplikacje bardzo rzadko wywołują bezpośrednio funkcje Kernela, ponieważ są dostępne znacznie wygodniejsze w użyciu funkcje wyższego poziomu.

- **Biblioteka systemowa** (system library) dostarcza podstawowego interfejsu między klientami a serwerami. Zawiera takie funkcje jak *Open()*, *Read()* i *Seek()*. Funkcje biblioteki systemowej w zależności od przekazanych im parametrów wysyłają odpowiednie komunikaty zgodne z formatem zdefiniowanym w *General Server Protocol*. Komunikaty te biblioteka systemowa przesyła następnie do odpowiedniego serwera i zwykle czeka na odpowiedź. Serwer odbiera komunikat, wykonuje żądane czynności i odsyła odpowiedź. Na poziomie biblioteki systemowej nie jest istotne, czy serwer i klient działają na tym samym, czy różnych procesorach. Ma to znaczenie na poziomie Kernela.

- **Biblioteka serwera** (server library) powstała, aby ułatwić tworzenie serwerów. Zawiera kod pozwalający przechowywać strukturę plików w pamięci, automatycznie obsługiwać wiele nadchodzących żądań i buforować dane.

- **Biblioteka narzędziowa** (utility library) zawiera funkcje, które powinny znaleźć się w Nucleusie, ale nie „pasowały" do poprzednich bibliotek. Są to m.in. funkcje gospodarki pamięcią, np. *memcpy()* i *strcpy()*.

- **Program ładujący** (loader) jest serwerem Heliosa, czyli programem, a nie biblioteką. Zajmuje się ładowaniem fragmentów kodu do procesorów i rozdziela kod pomiędzy procesorami, jeśli jest to możliwe. Kiedy w procesorze ma zostać uruchomiony program, tworzony jest plik w katalogu `/loader` tego procesora i pobierany jest kod programu, zwykle z dysku. Jeśli program używa bibliotek, których nie ma w danej chwili w pamięci, są one automatycznie doładowywane.

- **Zarządca procesów** (processor manager) udostępnia inny serwer Heliosa, `/tasks`. Pozwala on uruchamiać klientom programy w procesorze. Oczekuje na sygnały wysyłane do konkretnego zadania (np. SIGINT

wysyłany, gdy użytkownik naciska CTRL-C) i dba o zwolnienie wszystkich zasobów po zakończeniu wykonywania programu. Zarządca procesów wykonuje też inne zadania: odmierza czas (w przypadku braku zegara sprzętowego) i przechowuje tablicę nazw dla konkretnego procesora.

Wyżej opisane elementy są obecne w każdym Nucleusie Heliosa. Możliwe jest dołączenie do Nucleusa dodatkowych serwerów. Na przykład w systemie startowanym z ROM-u zamiast procesora `I/O` Nucleus powinien zawierać serwer plików, aby możliwe było dalsze ładowanie systemu z dysku.

Zarządca procesów wypełnia jeszcze jedno zadanie nie wymienione wcześniej. Po wystartowaniu systemu przez procesor `I/O` lub z ROM-u wykonuje program `/helios/lib/init`. Jest on następnym krokiem inicjalizacji systemu.

3 Interakcja z systemem - podstawy

3.1 Uruchamianie systemu

System operacyjny Helios działający w oparciu o nadrzędny komputer typu PC zwykle jest zainstalowany w katalogu `helios`. Z poziomu DOSa należy uczynić ten katalog katalogiem bieżącym i uruchomić program `server`. System powinien zgłosić się komunikatem podobnym do poniższego:

```
                                                    console


                    Helios Operating System
                           Version xx
            (C) Copyrigt 1987-90, Perihelion Software Ltd.


                                                     login:
```

Helios oczekuje na zalogowanie się użytkownika. Zwykle w każdym systemie dostępny jest użytkownik **guest**. Po wpisaniu nazwy użytkownika i potwierdzeniu jej klawiszem Enter należy podać jeszcze hasło, chociaż system można skonfigurować tak, aby hasła nie były wymagane.

3.2 Podstawowe operacje

Helios jest systemem zbliżonym w obsłudze do Unix-a. Użytkownicy znający ten system nie powinni mieć kłopotów z korzystaniem z Heliosa. Należy jednak zaznaczyć, że niektóre polecenia Unix-a nie są dostępne w Heliosie, a inne — mimo tej samej nazwy — mogą działać inaczej niż w Unix-ie.

Wykonywanie polecenia może być przerwane przez jednoczesne naciśnięcie CTRL i C. W najprostszym przypadku polecenie jest pojedynczym słowem i nie wymaga podawania parametrów:

```
% date
Date : Wed Apr 3 10:01:14 1996
%
```

Próba wydania polecenia nieistniejącego w systemie spowoduje wypisanie komunikatu

```
Command not found.
```

Uwaga: nazwy plików muszą być poprawnymi nazwami w systemie DOS.

W systemie Helios jest dostępny opis poszczególnych poleceń. Ma on postać prostego interakcyjnego programu z możliwością przeglądania wszystkich poleceń. Uruchomienie systemu pomocy następuje po wydaniu polecenia `help`, z nazwą polecenia jako opcjonalnym parametrem. System pomocy można opuścić naciskając klawisz **Q**.

Helios zapamiętuje wydawane ostatnio przez użytkownika polecenia. Można do nich wrócić naciskając klawisze kursora (góra, dół). Domyślnie pamiętanych jest 20 poleceń. Ustawienie to można zmienić modyfikując odpowiednio plik `cshrc`. Ostatnio wydane polecenie można powtórzyć wpisując dwa wykrzykniki. Listę poleceń można przejrzeć poleceniem `history`. Wybór pozycji z tej listy odbywa się przez wpisanie wykrzyknika i numeru pozycji.

Następujące polecenia działają zgodnie z oczekiwaniami:

`&`	uruchomienie zadania w tle
`alias`	nadawane poleceniu nazwy zastępczej
`cat`	łączenie plików
`cd`	zmiana katalogu
`cp`	kopiowanie plików
`diff`	szukanie różnic między plikami
`emacs`	edytor tektów
`exit`	wyjście z shella
`fgrep`	szukanie łańcuchów w tekście
`help`	pomoc (ze spisem poleceń i prostym menu)
`history`	lista ostatnio wydanych poleceń
`jobs`	lista zadań wykonywanych w tle
`kill`	usunięcie zadania wykonywanego w tle
`login`	zalogowanie nowego użytkownika
`logout`	wyjście z Heliosa
`ls`	lista plików w bieżącym lub podanym katalogu
`make`	kompilacja sterowana skryptem
`mkdir`	tworzenie katalogu
`more`	wypisywanie zawartości katalogu
`mv`	przenoszenie lub zmiana nazwy pliku
`pr`	wydruk pliku
`printenv`	lista zmiennych systemowych
`ps`	lista procesorów i zadań
`pwd`	wypisanie bieżącego katalogu
`remote`	uruchomienie programu na innym procesorze
`rm`	usuwanie plików i katalogów
`set`	ustawianie zmiennych shella
`setenv`	ustawianie zmiennych środowiskowych
`shell`	uruchomienie nowego shella
`source`	skrypty shella
`unalias`	usuwanie nazwy zastępczej
`unset`	usuwanie zmiennej shella
`unsetenv`	usuwanie zmiennej środowiskowej
`wsh`	uruchomienie shella okienkowego

4

4 Tworzenie oprogramowania w systemie Helios

4.0.1 Program `make`

Program ten ułatwia kompilację i łączenie programów składających się z wielu części. Rozważmy program w języku C składający się z dwóch modułów (kompilator zostanie dokładniej opisany później):

main.c

```
#include <stdlib.h>
extern void say_hello(void);
int main(int argc, char **argv)
{
say_hello();
return (EXIT_SUCCESS);
}
```

io.c

```
#include <stdio.h>
void say_hello()
{
puts("Hello world.");
}
```

Najprostszą metodą kompilacji tych modułów jest podanie kompilatorowi nazw obu plików:

```
c -o hello main.c io.c
```

Kompilator skompiluje **main.c**, **io.c** i utworzy plik wykonywalny **hello**. Jest to proste rozwiązanie, ale czasochłonne, gdy zajdzie konieczność kompilacji po zmianie dokonanej w jednym tylko module. Uruchomienie kompilatora w wyżej opisany sposób spowoduje powtórną kompilację wszystkich, a nie tylko zmienionego, modułów. Pewnym rozwiązaniem jest wykorzystanie pośrednich plików tworzonych przez kompilator. Mają one rozszerzenie **.o**. Wydanie polecenia

```
c -o hello main.o io.o
```

spowoduje jedynie łączenie wskazanych modułów i utworzenie programu **hello**. Kompilacja pliku źródłowego do pliku **.o** może być przeprowadzona przez wykorzystanie opcji **-c**:

```
c -c main.c
```

Nie jest podejmowana wtedy próba stworzenia programu wykonywalnego, która skończyłaby się błędem z powodu braku definicji funkcji *say_hello()*.

Innym rozwiązaniem jest wykorzystanie programu **make**. Rozważmy następujący przykład: program **teatime** składa się z plików **assam.c**, **water.c**, **sugar.c**, **cream.c**, **scones.c** i **jam.c**. Dodatkowo mamy plik nagłówkowy **teapot.h**, który dołączany jest przez **assam.c** i **water.c**. Narzędzie **make** czyta plik o nazwie **makefile** zawierający opis czynności koniecznych do utworzenia pliku wykonywalnego. Zawartość tego pliku może być następująca:

```
teatime: assam.o water.o sugar.o cream.o scones.o
jam.o
c -o teatime assam.o water.o sugar.o cream.o scones.o
jam.o
assam.o: assam.c teapot.h
c -c assam.c
water.o: water.c teapot.h
c -c water.c
sugar.o: sugar.c
c -c sugar.c
cream.o: cream.c
c -c cream.c
scones.o: scones.c
c -c scones.c
jam.o: jam.c
c -c jam.c
```

Pierwszą pozycją w każdym rekordzie tego pliku jest **cel**, oznaczający obiekt, który ma utworzyć **make**. Cel jest zwykle, ale nie zawsze, plikiem.

Dalej następuje dwukropek i lista zależności. Zawarte są w niej obiekty, które muszą zostać utworzone wcześniej niż cel. W następnym wierszu, zaczynającym się od znaku tabulacji, zapisane jest polecenie tworzące cel. To polecenie zostanie w razie potrzeby wykonane przez program **make**. **Make** korzysta z daty i czasu utworzenia plików do ustalenia, czy konieczna jest ponowna kompilacja. Możliwe jest też korzystanie ze zmiennych. Na przykład pierwsza reguła opisuje listę modułów i wymienia ją w dwóch miejscach. Można zapis ten skrócić stosując zmienną `objects`:

```
objects = assam.o water.o sugar.o cream.o scones.o
jam.o

teatime: $(objects)
c -o teatime $(objects)
```

Istnieje też wiele użytecznych predefiniowanych zmiennych:

$@ nazwa celu,

dol strz nazwy obiektów z listy zależności,

$¡ nazwa pierwszego obiektu z listy zależności,

$* nazwa celu bez rozszerzenia,

$? nazwa obiektu z listy zależności, który spowodował konieczność ponownego utworzenia celu

W pliku **makefile** można też umieszczać komentarze, rozpoczynając wiersz od znaku #. Program **make** potrafi również czytać inne pliki niż **makefile**. Nazwę pliku można podać korzystając z opcji **-f**.

4.1 Edytor tekstowy MicroEmacs

W systemie Helios dostępny jest pełnoekranowy edytor tekstowy MicroEmacs, wywoływany komendą `em`. W tablicy 1 przedstawiono sekwencje klawiszy odpowiadające podstawowym operacjom dostępnym w tym edytorze.

Kombinacja klawiszy	Realizowana operacja
ESC ?	wywołanie okna podpowiedzi
CTRL-X CTRL-C	zakończenie pracy bez nagrywania edytowanego pliku
CTRL-X CTRL-S	nagranie edytowanego pliku
ESC Z	nagranie edytowanego pliku i zakończenie pracy
CTRL-S	szukanie wzorca w tekście „w przód"; wprowadzanie wzorca należy zakończyć naciśnięciem klawisza ESC
CTRL-R	szukanie wzorca w tekście „w tył"; wprowadzanie wzorca należy zakończyć naciśnięciem klawisza ESC
CTRL-X S	stopniowe wyszukiwanie wzorca „w przód"
CTRL-X R	stopniowe wyszukiwanie wzorca „w tył"
ESC-R	globalna zamiana ciągów znaków
CTRL-X 0	skasowanie bierzącego okna
CTRL-X 1	skasowanie wszystkich okien z wyjątkiem bierzącego
CTRL-X 2	podział bierzącego okna na dwie części
CTRL-X O	przejście do następnego okna
CTRL-X P	przejście do poprzedniego okna
CTRL-L	odświeżenie ekranu
CTRL-K	kasowanie znaków od pozycji kursora do końca bierzącej linii
CTRL-X CTRL-I	wstawienie zawartości nowego pliku
CTRL-X CTRL-R	wczytanie nowego pliku do bierzącego bufora
CTRL-X CTRL-B	wyświetlenie listy buforów
CTRL-X B	wybranie bufora
CTRL-X K	skasowanie bufora

Tablica 1: Podstawowe sekwencje sterujące dostępne w edytorze MicroEmacs

5 Programowanie równoległe w systemie Helios

5.1 Model obliczeń

Na projekt systemu silny wpływ wywarła notacja CSP (ang. communicating sequential processes) [2]. Przyjętym modelem obliczeń jest zbiór asynchronicznie wykonujących się procesów współbieżnych, które komunikują się między sobą przy pomocy przesyłanych wiadomości (ang. message). Przesyłanie wiadomości nie musi odbywać się w sposób synchroniczny, co jest rozszerzeniem w stosunku do notacji CSP.

5.2 Zadania

Elementy składowe programu równoległego nazywane są zadaniami (ang. task). Zadaniem jest program sekwencyjny jedno- lub wielowątkowy (ang. multithreaded) napisany w dowolnym języku programowania. Każde zadanie jest kompilowane i linkowane osobno, podobnie jak to ma miejsce w przypadku procesów w systemie Unix. W Heliosie wszystkie wątki jednego zadania wykonują się w tym samym procesorze. Liczba zadań oraz topologia połaczeń między nimi nie jest uzależniona od liczby procesorów oraz sposobu ich połączenia w sieci, w której program równoległy ma być wykonywany. Rozmieszczenie zadań w sieci procesorów dokonywane jest automatycznie przez system operacyjny. Programista może także jawnie przypisać konkretnym procesorom zadania, które mają być w nich wykonywane. W momencie startu programu równoległego Helios tworzy oraz otwiera na poziomie biblioteki zgodnej ze standardem Posix wszystkie łącza służące do komunikacji między zadaniami wyspecyfikowane w skrypcie CDL opisującym ten program.

5.3 Komunikacja między zadaniami w systemie Helios

W systemie Helios dostępne są trzy sposoby przekazywania informacji między zadaniami: łącza (ang. pipe), gniazda (ang. socket) oraz przesyłanie komunikatów (ang. message passing).

Łącza są najczęściej stosowanym środkiem komunikacji między zadaniami. Zapewniają one automatyczne wykrywanie i obsługiwanie takich zdarzeń jak zagubienie komunikatu (np. wskutek przepełnienia buforów komunikacyjnych) lub uszkodzenie scieżki przesyłowej (np. wskutek awarii jednego z procesorów). Odwołanie do łączy w zadaniach napisanych w języku C dokonuje się poprzez deskryptory plików, a ich obsługa — przy pomocy funkcji *write*() oraz *read*(). Komunikacja przy pomocy łączy jest quasi-synchroniczna. Wykonywanie funkcji *write*() w zadaniu-nadawcy kończy się w chwili przesłania całego komunikatu, a wykonanie funkcji *read*() w zadaniu-odbiorcy powoduje odebranie takiej części komunikatu, jaka jest w danej chwili dostępna. Może zdarzyć się, że zadanie-odbiorca otrzyma komunikat nie w całości, lecz podzielony na mniejsze fragmenty. Gwarantuje się jednak, że ich względne uporządkowanie będzie identyczne, jak w oryginalnym komunikacie. Przy pomocy jednego łącza mogą komunikować się tylko dwa zadania, które mogą znajdować się w dowolnym miejscu sieci (np. na różnych, oddalonych od siebie procesorach). Możliwe jest wykorzystanie jednego łącza do realizacji komunikacji dwustronnej, choć nie jest to zalecane, gdyż wymaga to szczególnej uwagi ze strony programisty i jest okazją do popełniania trudnych do wykrycia i usunięcia błedów. Łącza doskonale nadają się do realizowania statycznych połączeń między zadaniami. Połaczenia te można określić w pliku konfiguracyjnym (skrypcie w języku CDL). Łącza trudno wykorzystać do tworzenia połączeń dynamicznych, za wyjątkiem komunikacji między procesem „rodzicem" (ang. parent process) a jego procesem potomnym (ang. child process).

Gniazdka są zalecanym sposobem dynamicznego nawiązywania połączeń między zadaniami. Mechanizm ten jest bardziej uniwersalny niż łącza, gdyż umożliwia wymianę danych między zadaniami wykonywanymi w różnych komputerach. Komunikacja przy pomocy gniazdek nie jest symetryczna.

Przesyłanie komunikatów jest mechanizmem najniższego poziomu, przy pomocy którego implementowane są łącza i gniazda. Bezpośrednie korzystanie z niego nie jest zalecane.

5.4 Język CDL

Język CDL (ang. Component Distribution Language) umożliwia wyspecyfikowanie zadań wchodzących w skład programu równoległego, ich wymagań (np. rozmiaru pamięci operacyjnej, typu procesora) i rozmieszczenia w sieci

procesorów oraz określenie połaczeń statycznych między zadaniami.

5.4.1 Deklaracja własności zadań

Przykładowa deklaracja własności zadania przedstawiona została poniżej.

component *nadawca* { **processor** T800;
memory 50000;
code `/helios/users/jbond/spy`;
streams , `>|` *out*;
puid 01;}
component *odbiorca* { **streams** `<|` *out*;}

Przeznaczenie każdej z użytych powyżej dyrektyw jest następujące:

processor służy do określenia typu procesora, w jakim dane zadanie ma być wykonywane. Zadanie *nadawca* musi być wykonywane w transputerze T800, zadanie *odbiorca* — w dowolnym;

memory służy do określenia minimalnego rozmiaru pamięci operacyjnej dostępnej dla procesora, w którym wykonywane będzie dane zadanie. Zadanie *nadawca* wymaga co najmniej 50000 bajtów;

code umożliwia określenie pliku, zawierającego wykonywalny kod danego zadania. Kod zadania *nadawca* znajduje się w pliku `/helios/users/jbond/spy`; kod drugiego zadania znajduje się w pliku `odbiorca` w bierzącym katalogu;

streams służy do określenia nazwanych połączeń statycznych między zadaniami. Kolejne pozycje na liście — oddzielone przecinkami — odpowiadają strumieniom udostępnianym zadaniu. Pierwszy element listy odpowiada strumieniowi o deskrytporze 0, drugi — strumieniowi o deskryptorze 1, itd. Symbol `>|` oznacza, że dane łącze jest łączem wyjściowym (służy do wysyłania danych przez zadanie), a symbol `<|` określa łącze, z którego zadanie będzie pobierać dane. Zadanie *nadawca* będzie korzystać tylko z jednego nazwanego łącza, którym jest łącze wyjściowe *out* o deskryptorze 1. Zadanie *odbiorca* będzie pobierać dane z tego samego łącza, posługując się deskryptorem 0;

puid umożliwia podanie identyfikatora (nazwy) procesora, w którym dane zadanie ma być wykonywane. Nazwy nadawane są procesorom w trakcie startu systemu.

Wszystkie dyrektywy są opcjonalne.

5.4.2 Deklaracja połaczeń między zadaniami

W punkcie 5.4.1 przedstawiono jeden sposób definiowania połączeń statycznych między zadaniami przy pomocy łączy nazwanych wprowadzanych dyrektywą **streams**. Metoda ta nie wystarcza do opisania wszystkich możliwych topologii połączeń, może być także niewygodna w użyciu. W języku CDL dostępne są cztery konstruktory (|, <>, ^^, |||) służące do specyfikowania połączeń statycznych między zadaniami przy pomocy nazwanych i nienazwanych łączy.

Konstruktor | służy do definiowania jednokierunkowego łącza między dwoma zadaniami (patrz rys. 3.a). Konstruktor <> umożliwia definiowanie dwukierunkowej komunikacji między dwoma zadaniami, która jest realizowana przy pomocy pary jednokierunkowych łącz (rys 3.b). Konstruktor ^^ pozwala na zdefiniowanie braku komunikacji między zadaniami (3.c)[4]. Konstruktor ||| używany jest głównie wraz z replikatorami, o których mowa w punkcie 5.4.3 i służy do specyfikowania tzw. farm (rys. 3.d).

Oprócz wymienionych wyżej konstruktorów można stosować także standardowe symbole przekierowania strumieni (<, >, >>). Do specyfikowania nazwanych łącz stosuje się symbole <| oraz >|, wspomniane w punkcie 5.4.1.

5.4.3 Replikatory

Replikatory ułatwiają tworzenie specyfikacji programów równoległych o wielu zadaniach składowych tego samego rodzaju. Farma przedstawiona na rysunku 3 jest nietypowa, gdyż posiada tylko jedno zadanie robocze B. Zazwyczaj farma składa się z wielu takich zadań, np. 10. Wypisywanie ich wszystkich po kolei byłoby uciążliwe. Zastosowanie replikatorów pozwala na ominięcie tego problemu. Poniższy przykład:

A [10] ||| B

[4]Komunikacja między tymi zadaniami może być nawiązana w inny sposób.

jest specyfikacją programu równoległego, który jest farmą 10 procesów roboczych B i jednego procesu-nadzorcy A. Ten sam program można opisać stosując inny rodzaj replikatorów, tzw. replikatory postfixowe, jak ma to miejsce w następującym przykładzie:

A <> (||| [10] B)

5.5 Kompilator języka CDL

Kompilator języka CDL wywoływany jest poleceniem

`cdl <opcje> <plik_we> <argumenty>`

Kompilator akceptuje następujące opcje:

-n plik wejściowy nie jest kompilowany, a jedynie sprawdzana jest jego składnia;

-c plik wejściowy jest kompilowany, ale nie jest wykonywany; powstały kod wynikowy wypisywany jest do strumienia **stdout**;

-o <plik_wy> podobnie jak opcja -c, z tym że powstały kod wynikowy umieszczany jest we wskazanym pliku;

-i wysłanie do strumienia **stderr** informacji o zadaniach i łączach, przy pomocy których te zadania komunikują się ;

-l <plik_wy> podobnie jak opcja -i, z tym że informacja zapisywana jest we wskazanym pliku.

Domyślnie kompilator pobiera kod źródłowy ze standardowego wejścia, kompiluje go i wykonuje.

6 Zadania

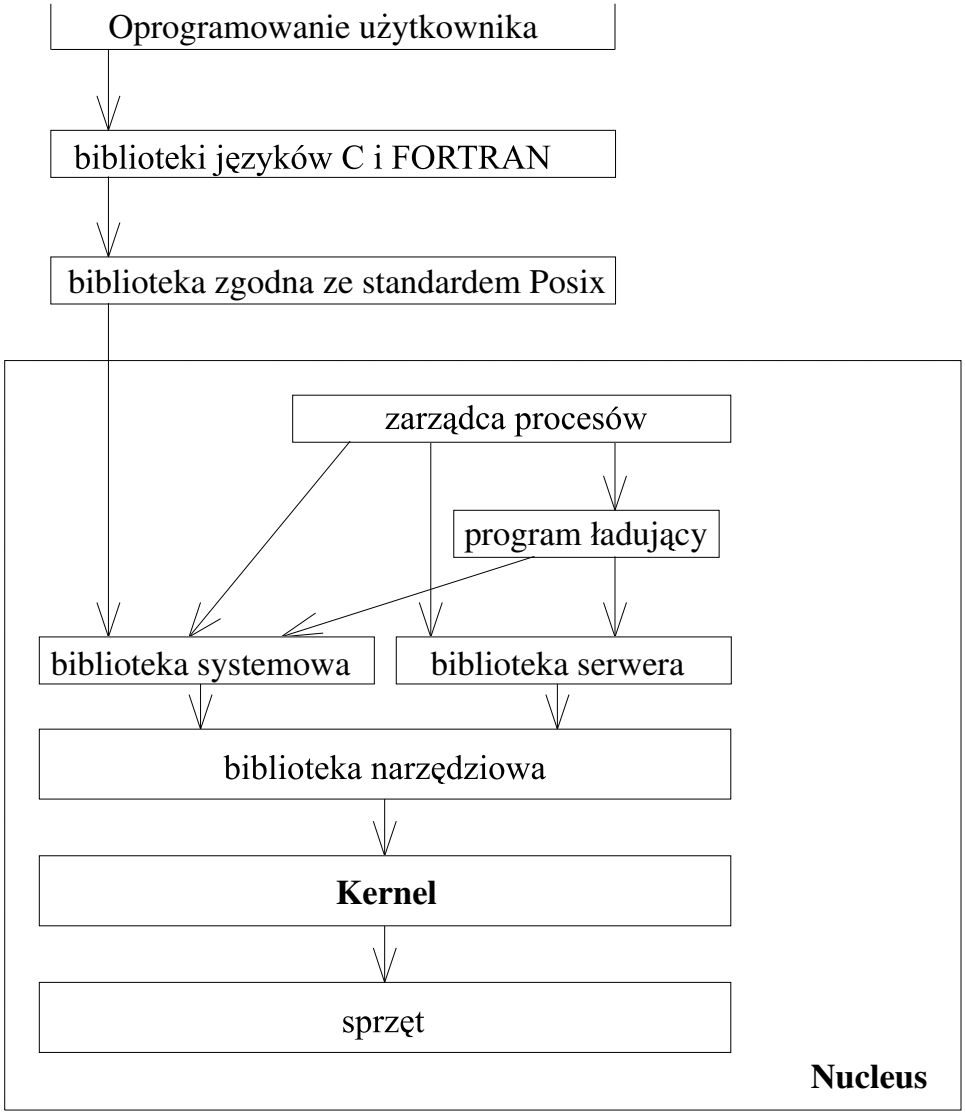
1. Wykonać kilka komend: ps, ls, domain, loaded, network.
2. Zapoznać się z programami przykładowymi wskazanymi przez prowadzącego.
3. Zaprogramować zadanie 5.3.8 z pozycji [3] (centrala telefoniczna). Porównać z rozwiązaniem w języku occam.

Cytowana literatura

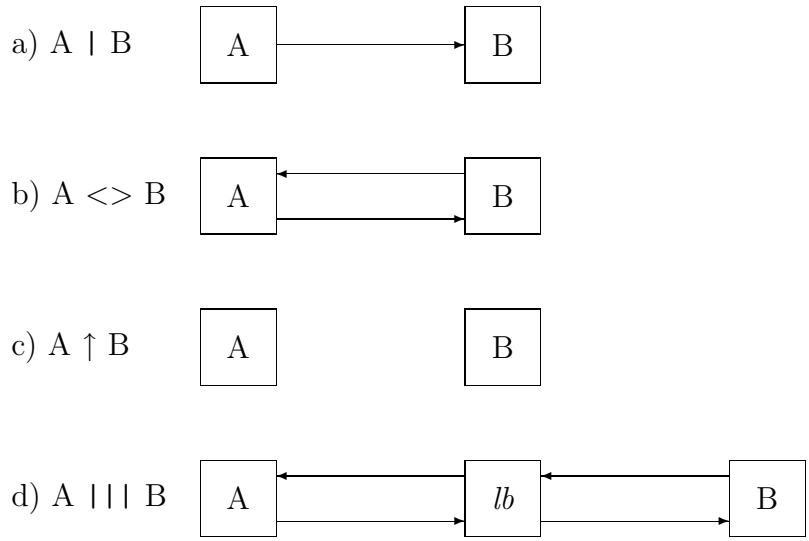
[1] *The Helios Parallel Operating System*, Perihelion Software Ltd, Prentice Hall, New York 1991.

[2] Hoare, C.A.R.: Communicating sequential processes, w: Communications of the ACM, Vol. 21, No. 8, pp. 666–677.

[3] Weiss, Z., Gruźlewski T.: Programowanie współbieżne i rozproszone, WNT, Warszawa 1993.



Rysunek 2: Budowa Nucleusa



Rysunek 3: Przykłady użycia konstruktorów definiujących połączenia między zadaniami: a) jednokierunkowe łącze; b) komunikacja dwukierunkowa; c) brak komunikacji; d) farma. Wielkimi literami (A, B) oznaczono zadania. Symbol *lb* oznacza zadanie równoważące obciążenie pracą zadań roboczych farmy.