Komunikacja międzyprocesowa i międzywątkowa

Moduł 1

procesy, wątki, synchronizacja

mgr inż. Tomasz Pawelec

Prowadzący

Tomasz Pawelec

mgr inż.

Software Consultant w Globallogic

12 lat doświadczenia zawodowego

- software consultant / software architect
  - systemy wbudowane / morskie / telekomunikace / automotive
  - C/C++

Wykładowca w Wyższej Szkole Techniczno-Ekonomicznej w Szczecinie

- Wprowadzenie do programowania w języku C pod systemy wbudowane
- Biblioteka standardowa C++
- Projekt praktyczny

Wykładowca w Zachodniopomorskim Uniwersytecie Technologicznym w Szczecinie

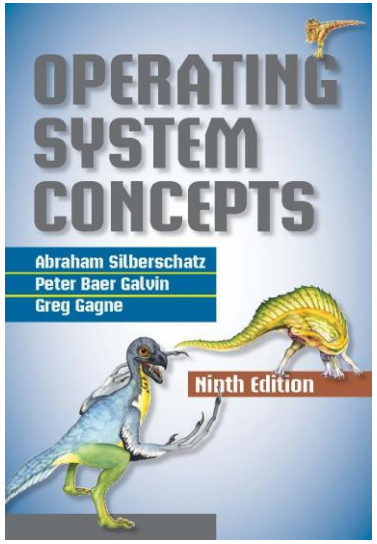
- Komunikacja międzyprocesowa i międzywątkowa

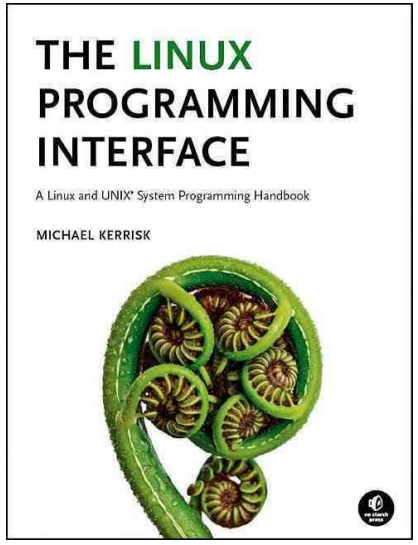
tomasz.pawelec@globallogic.com

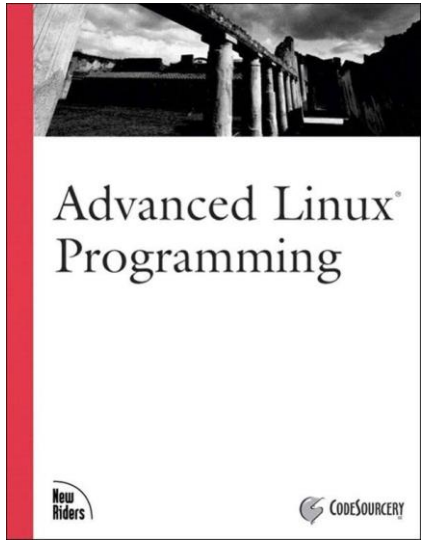
tpawelec.pl@gmail.com (preferowany adres)

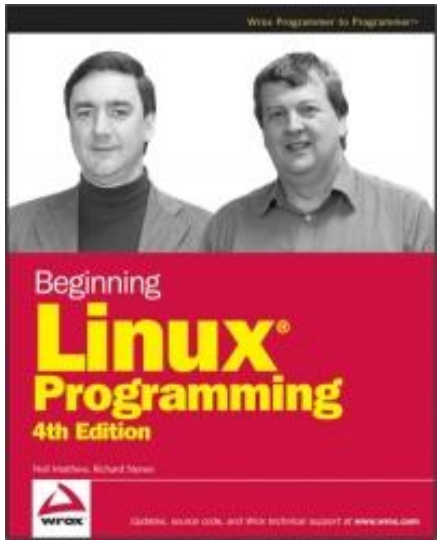
Literatura kursu

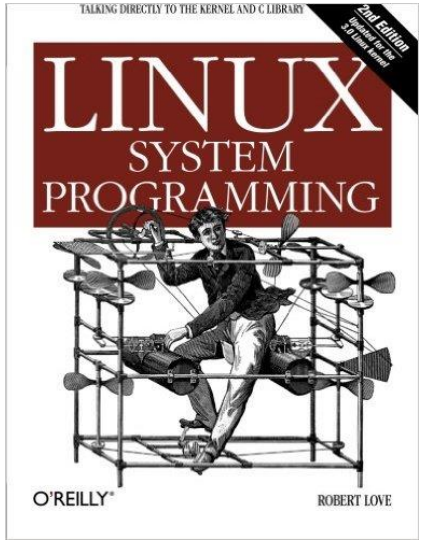
1. „The Linux programming Interface”, Michael Kerrisk
2. „Operating System Concept”, Abraham Silberschatz, Peter Galvin Greg Gagne
3. „Beginning Linux Programming”, Neil Matthew Richard Stones
4. „Advanced Linux Programming”, Mark Mitchell, Jeffrey Oldham, Alex Samuel
5. „Linux System Programming”, Robert Love
6. „C++ Concurrency In Action”, Anthony Williams

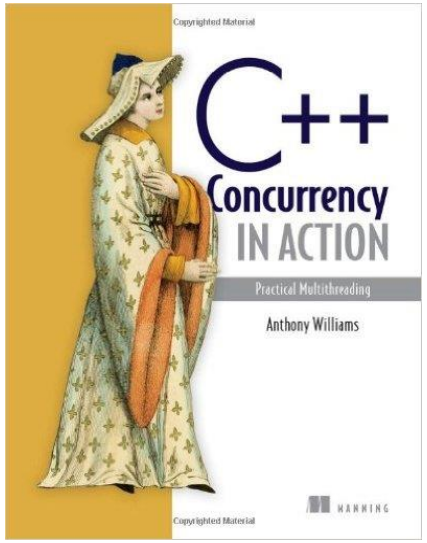












Linux man (manual):
http://man7.org/linux/man-pages/index.html

Zakres kursu

Moduł 1: [4h] Procesy, wątki, synchronizacja 1

Moduł 2: [4h] Synchronizacja 2, komunikacja międzyprocesowa

Moduł 3: [2h] Równoległość w C++

Moduł 4: [2h] Zaliczenie

Organizacja kursu

(1) Kontakt z prowadzącym / konsultacje
- kontakt mailowy
- Przed zajęciami / w trakcjie przerw

(2) Zadania domowe (dwa)
- ostateczny termin oddania zadań na tydzień przed ostatnimi zajęciami

(3) Jedna, ta sama ocena za laboratoria/ćwiczenia/warsztaty oraz wykład

(4) Zaliczenie – obecność na zajęciach

(5) Ocena wyżej:
- zaliczenie na ostatnich zajęciach (test jednokrotnego wyboru)
- zadania domowe

Moduł 1 - agenda

1. Mars Pathfinder
2. Procesy
3. Wątki
4. Wątki – sychronizacja i komunikacja
5. Zadanie

Mars Pathfinder

Zródła:

https://en.wikipedia.org/wiki/Mars_Pathfinder

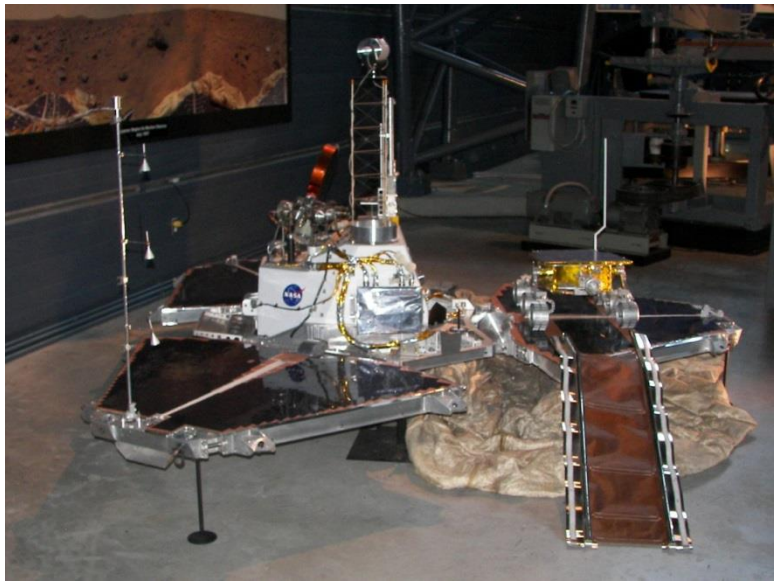
Mars Pathfinder

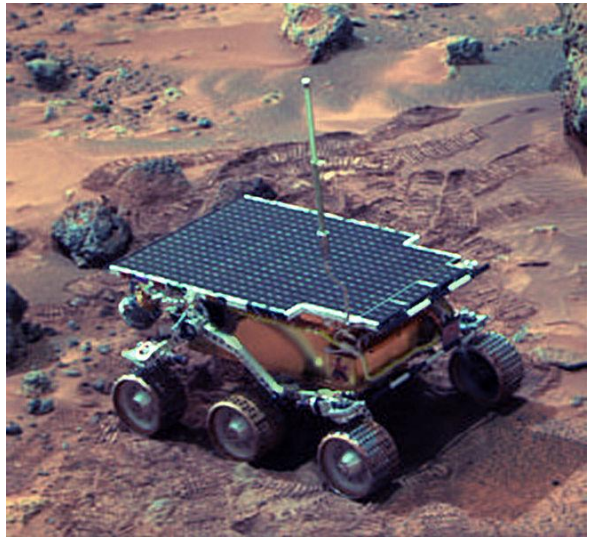
- rozpoczęty w 1993
- kosztował 265 milonów dolarów
- wystrzelenie 4.10.1996
- wylądowanie na Marsie 4.07.1997
- VxWorks

- lądownik (ang. lander) + trzy łaziki(ang. rover)

Cele:
- sprawdzić nowy sposób lądowania
- zebrać zdjęcia i próbki gleby
- sprawdzić autonomiczne łaziki

Po wylądowanianiu system
zaczął resetować się

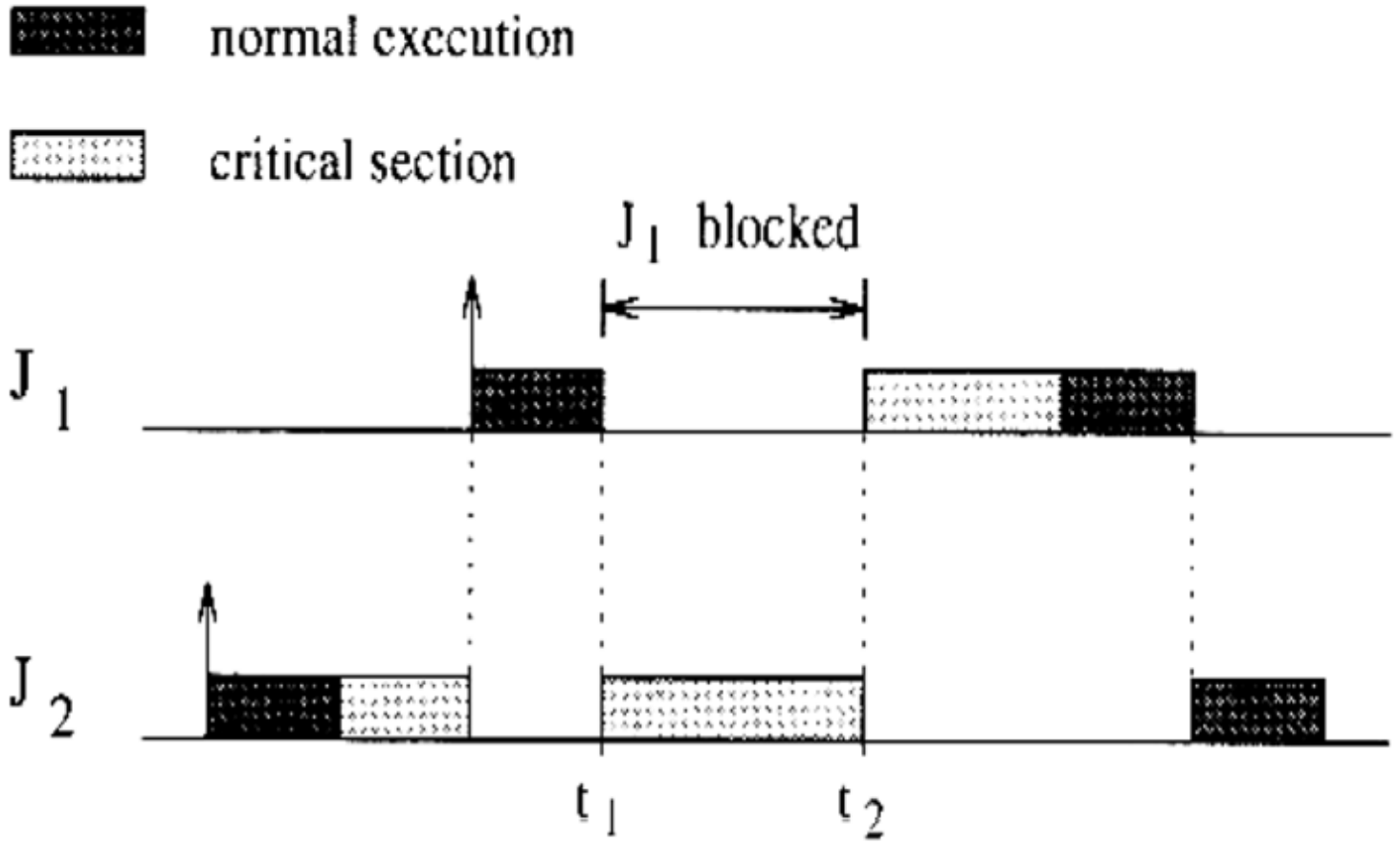




Zródła:
https://en.wikipedia.org/wiki/Mars_Pathfinder

Mars Pathfinder

- prio(J1) > prio(J2)
- zjawisko „Nieuniknionego blokowania” (ang. ***Unavoidable blocking***) – wątek z wyższym priorytetem jest zablokowany z powodu oczekiwanie na dostęp do sekcji krytycznej

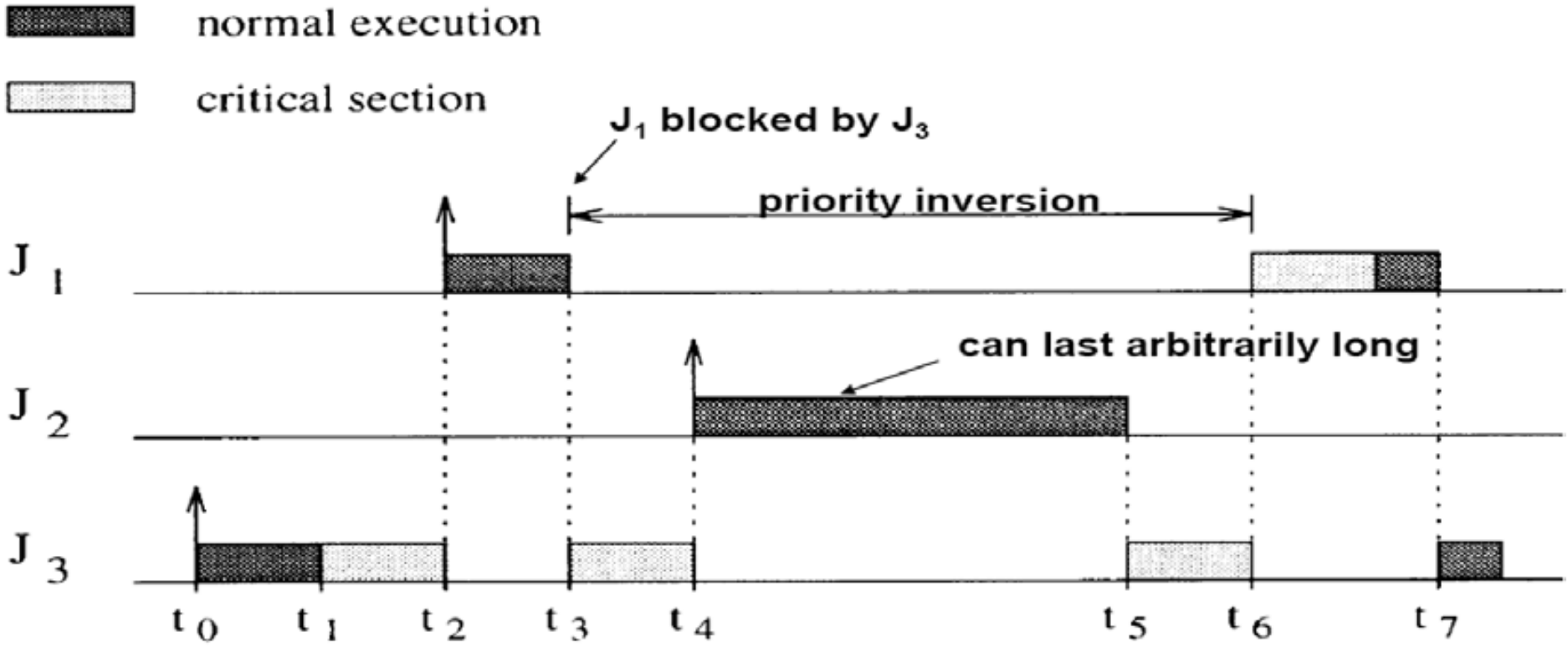


Zródła:

http://www.cse.chalmers.se/~risat/Report_MarsPathFinder.pdf

Mars Pathfinder

- prio(J1) > prio(J2) > prio(J3)
- wydaje sie, ze J2 ma największy priorytet ponieważ blokuje obydwa wątki – problem zamiany priorytetów (ang. ***Priority inversion problem***)

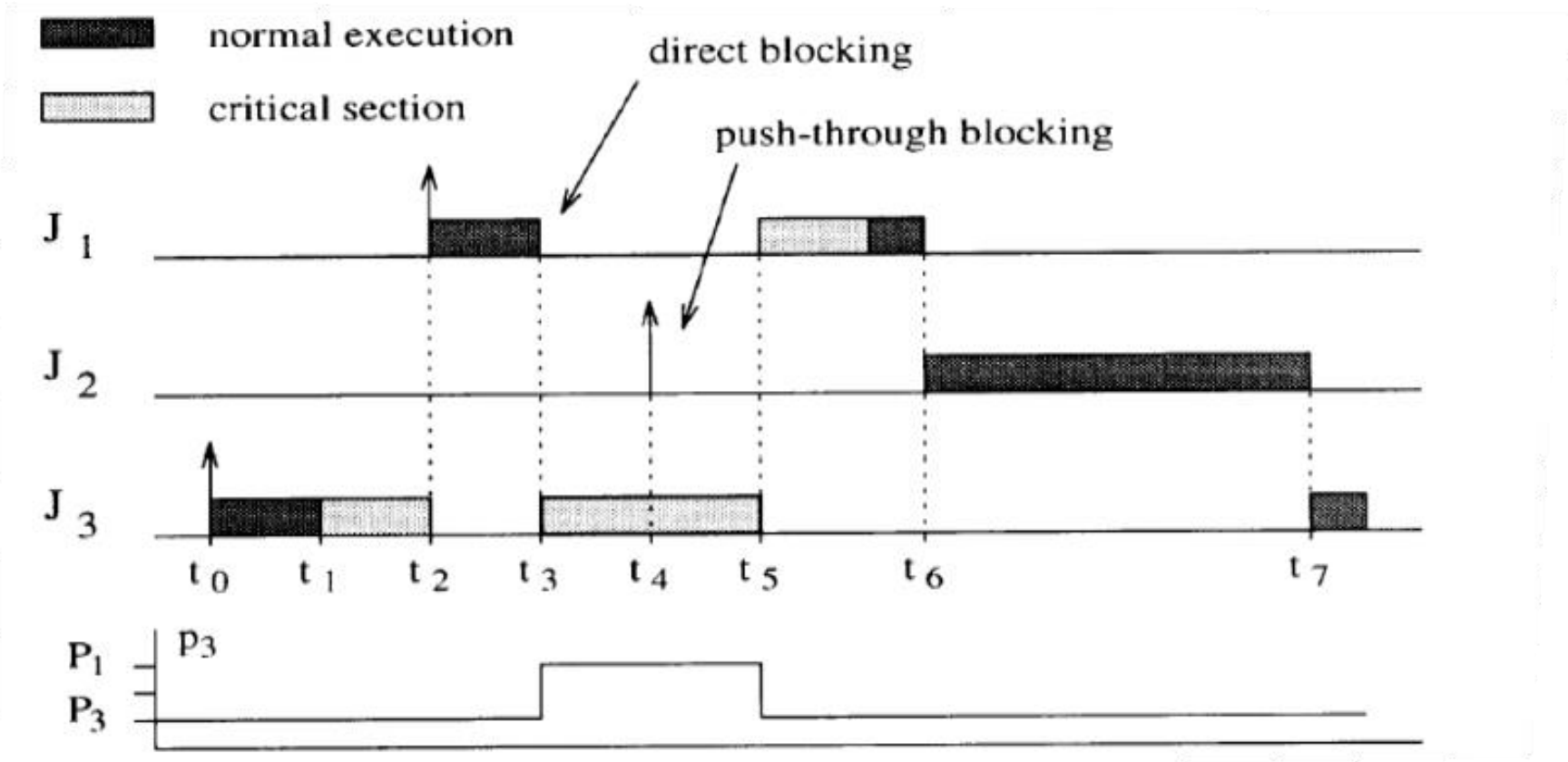


Zródła:

http://www.cse.chalmers.se/~risat/Report_MarsPathFinder.pdf

Mars Pathfinder

- „Protokół dziedziczenia priorytetu” (ang. ***Protocol inheritance protocol***)
- wątek blokujący dostęp do współdzielonego zasobu (J3) czasowo dostaje/dziedziczy najwyższy priorytet ze wszystkich zablokowanych wątków (J1)



Zródła:

http://www.cse.chalmers.se/~risat/Report_MarsPathFinder.pdf

Mars Pathfinder

Trzy zadania:

- **J1 - High** (szyna komunikacyjna, mutex)

- **J2 - Medium** (zapisywał dane do FLSH)
  - nie było już miejsca i czekał aż się zwolni => błąd
  - blokował wątki J1 oraz J3
  - blokada ta była zauważona przez watchdog, który powodował reset systemu

- **J3 - Low** (szyna komunikacyjna, mutex)

Zródła:

http://www.cse.chalmers.se/~risat/Report_MarsPathFinder.pdf

Mars Pathfinder

Trzy zadania:

- **J1 - High** (szyna komunikacyjna, mutex)

- **J2 - Medium** (zapisywał dane do FLSH)
  - nie było już miejsca i czekał aż się zwolni => błąd
  - blokował wątki J1 oraz J3
  - blokada ta była zauważona przez watchdog, który powodował reset systemu

- **J3 - Low** (szyna komunikacyjna, mutex)

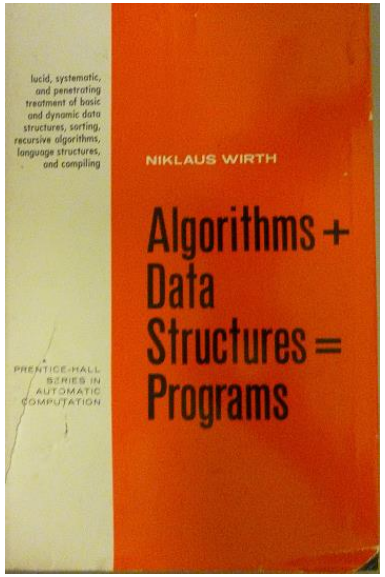
Rozwiązanie:

- VxWorks – opcja „**inversion priority**” była wyłączona, trzeba było ją włączyć

Zródła:
https://en.wikipedia.org/wiki/Mars_Pathfinder
https://www.youtube.com/watch?v=lyx7kARrGeM

Procesy

Podstawowe pojęcia



Program (ang. *Program*)
algorytmy + struktury danych

Proces (ang. *Process*)
instancja uruchomionego programu

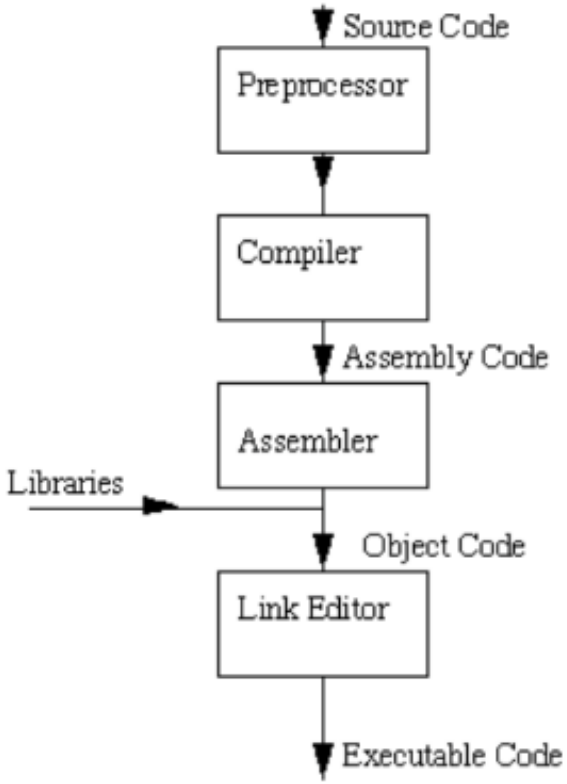
Wątek (ang. *Thread*)
część programu wykonywana współbierznie w obrębie jednego procesu

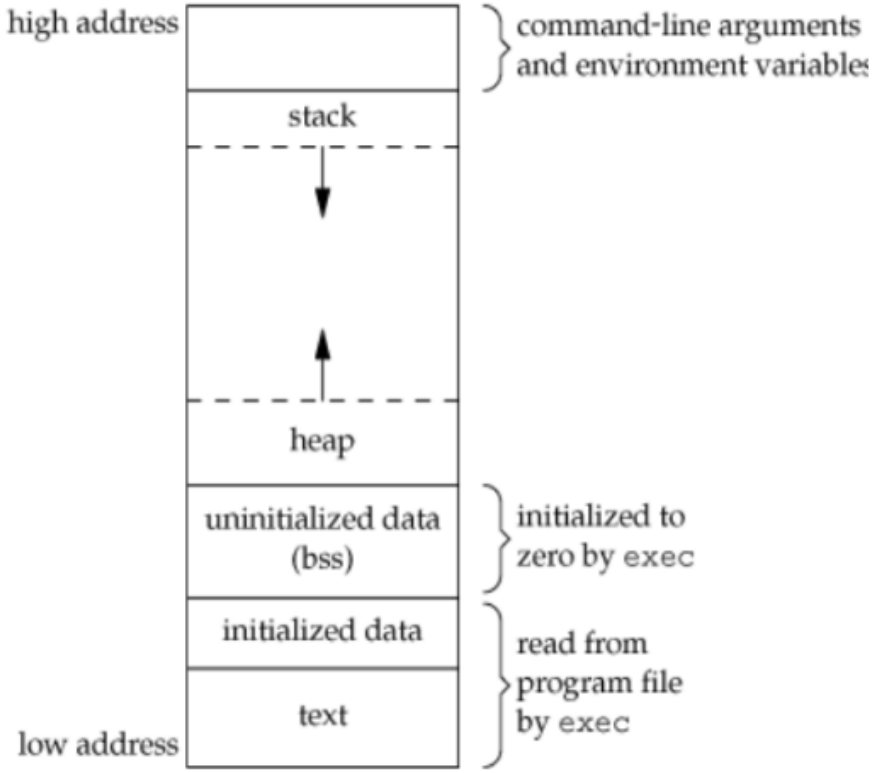
System operacyjny (ang. *Operating System*)
„System operacyjny jest programem, który działa jako pośrednik między użytkownikiem komputera a sprzętem komputerowym” *Abraham Silberschatz*

Niektóra zadania systemu operacyjnego:
- definiowanie interfejsu użytkownika
- udostępnianie systemu plików (zarządzania pamięcią)
- sterowanie urzędzeniami wejścia-wyjścia
- ***udostępnianie środowiska do wykonywania programów***

Zródła:
https://www.cs.cf.ac.uk/Dave/C/node3.html
http://www.tenouk.com/ModuleW.html

Kompilacja i uruchomienie programu



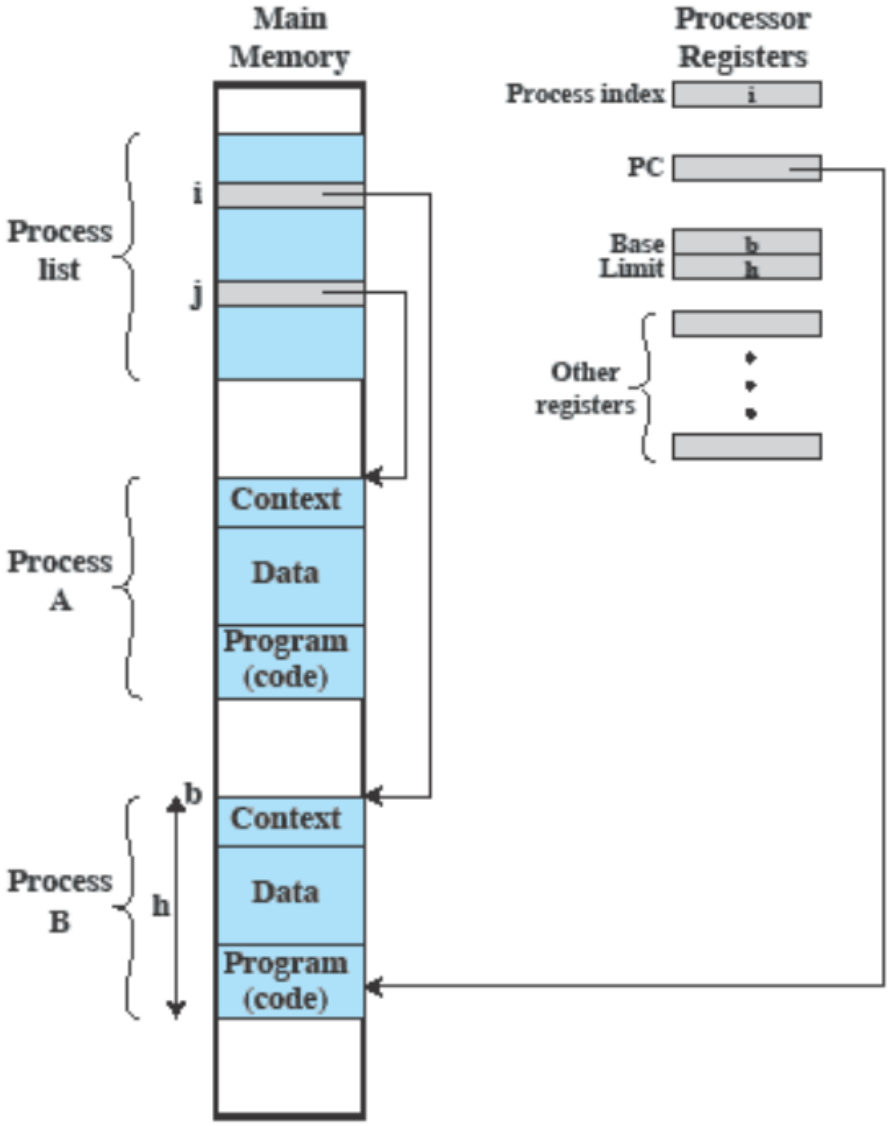


Zródła:
https://www.cs.cf.ac.uk/Dave/C/node3.html
http://www.tenouk.com/ModuleW.html

Pojęcie procesu

Proces jest egzemplarzem wykonywanego programu posiadającym:

- zasoby systemowe (ang. ***system resources***)
- pamięć (ang. ***memory***)
- atrybuty bezpieczeństwa (ang. ***security attributes***) np: właściciej, prawa dostępu
- stan



Zródła:

http://cs.nyu.edu/courses/fall12/CSCI-GA.2250-001/slides/Chapter02.pdf

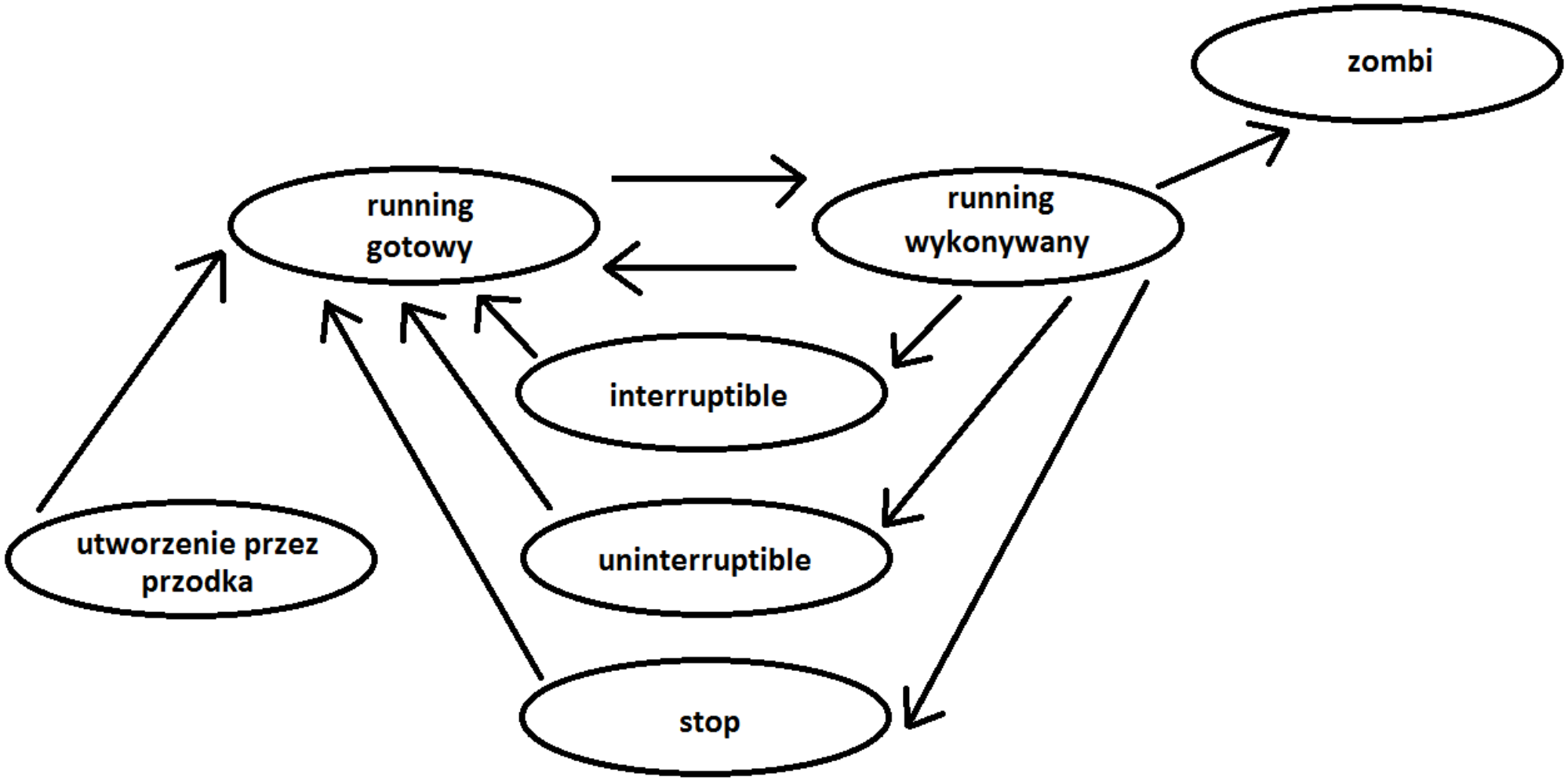
Proces - stan

PROCESS STATE CODES

R running or runnable (on run queue)
D uninterruptible sleep (usually IO)
S interruptible sleep (waiting for an event to complete)
Z defunct/zombie, terminated but not reaped by its parent
T stopped, either by a job control signal or because
it is being traced

```
< high-priority (not nice to other users)
N low-priority (nice to other users)
L has pages locked into memory (for real-time and custom IO)
s is a session leader
l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
+ is in the foreground process group
```

Proces – stan


zombi
running
gotowy
running
wykonywany
interruptible
utworzenie przez
przodka
uninterruptible
stop

Planista (ang. *Scheduler*)

Część systemu operacyjnego odpowiedzialnego za:

- wybór procesu do uruchomienia
- usunięcie uruchomionego process

Przełączanie kontekstów (ang. ***Contex swtiching***):

- kontekst procesu jest reprezentowany przez blok kontrolny procesu (ang. ***Process Control Block***, PCB)
- stan procesu wywłaszczanego musi być zachowany
- **przełącznie kontekstów jest czasochłonne !!!**
  - można wykorzystać wątki

Równoległość (ang. parallelism / concurrency)

Multiple processes executing during the same period of time:

Parallel computing is a type of computation in which many calculations or the execution of process are carried out **simultaneously**

- goal of speeding up ***computations***
- execution occurs at the same physical instant: for example, on **separate processor** of a multi-processor machine
- impossible on a (on-core) single processor, as only one computation can occur at any instant (during any single clock cycle)
- recommendation for C++ => **no raw synchronization primitives** (threads, mutexes, etc)[2]

Concurrent computing is a type of computation in which several computations are executed during **overlapping time periods** — concurrently — instead of sequentially (one completing before the next starts:

- the goal here is to ***model processes in the outside world that happen concurrently***, such as multiple clients accessing a server at the same time
- consists of process lifetimes overlapping, but execution need not happen at the same instant

Zródła:
[1] https://en.wikipedia.org/wiki/Concurrent_computing
[2] https://www.youtube.com/watch?v=4OCUEgSNIAY&list=WL&index=15&t=0s

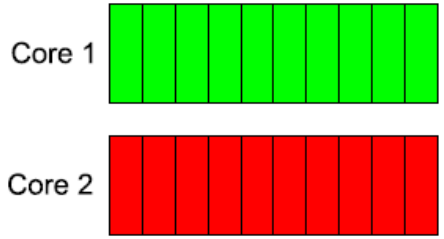
Równoległość (ang. parallelism / concurrency)

For example, given two tasks, T1 and T2:

- T1 may be executed and finished before T2 or vice versa (**sequential**)

- T1 and T2 may be executed alternately (**concurrent, task switching**)

- T1 and T2 may be executed simultaneously at the same instant of time (**parallel**)



Proces – podstawowe polecenia

Lista procesów

```
#top
```

Procesy skojarzone z użytkownikiem

```
#ps
```

Wszystkie procesy

```
#ps –aux | less
a – dla wszystkich użytkników
u – szczczegółowe informacje
x – demony
```

Wszystkie procesy -sortowanie

```
#ps –aux --sort=-pcpu | less
```

Proces – podstawowe polecenia

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.1 119564  5776 ?        Ss   15:41   0:02 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    15:41   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    15:41   0:00 [ksoftirqd/0]
root           5  0.0  0.0      0     0 ?        S<   15:41   0:00 [kworker/0:0H]
root           7  0.0  0.0      0     0 ?        S    15:41   0:00 [rcu_sched]
```

USER user owning the process
PID process ID of the process
%CPU It is the CPU time used divided by the time the process has been running.
%MEM ratio of the process’s resident set size to the physical memory on the machine
VSZ virtual memory usage of entire process (in KiB)
RSS resident set size, the non-swapped physical memory that a task has used (in KiB)
TTY controlling tty (terminal)
STAT multi-character process state
START starting time or date of the process
TIME cumulative CPU time
COMMAND command with all its arguments

Operacje na procesach

Tworzenie (ang. ***process creation***)

- ***exec(3)***
  - ***execl(3)***
  - ***execlp(3)***
  - ***execle(3)***
  - ***execv(3)***
  - ***execvp(3)***
  - ***execvpe(3)***
- ***system(3)***
- ***fork(2)***

Kończenie (ang. ***process termination***)

- poprawne (ang. ***normal***)
  - zakończenie funkcji ***main()***
  - ***exit(2), _exit(2)***
- niepoprawne (ang. ***abnormal***)
  - ***abort(3)*** - sami
  - ***kill(2)*** - inn
- ***wait(2),waitpid(2)***
- zombie, orphan, deamon

Komunikacja (ang. ***interprocess communication***, IPC) => moduł 2

Operacje na procesach - tworzenie

Funkcja ***exec(3)***

- zastąpienie aktualnie wykonywanego procesu kodem innego programu
- nie porwraca się do procesu wywołującego
- path – program, który będzie zastępował aktualny process
- arg – argv0, argv1, ... , argvn

```
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ...);

int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],char *const envp[]);
```

Zródła:
https://en.wikipedia.org/wiki/Exec_(system_call)

Operacje na procesach - tworzenie

Funkcja ***system(3)***

- tworzy nowy proces przez wywołanie funkcji ***fork()*** w którym to następnie wywoływane jest polecenia shella
- po wykonaniu polecenia wracamy do programu

int system(const char *command);

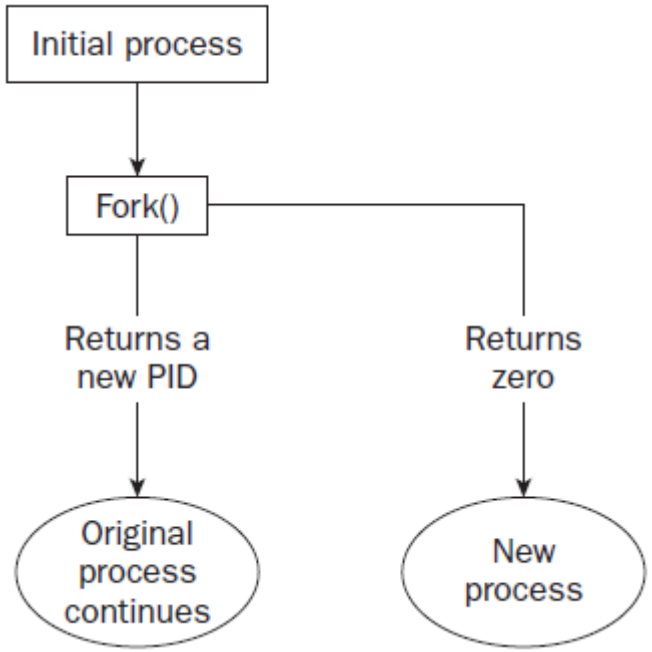
Zródła:
https://en.wikipedia.org/wiki/Exec_(system_call)

Operacje na procesach - tworzenie

Funkcja ***fork(2)***

- tworzy nowy proces (ang. ***child***) będący kopią procesu rodzica (ang. ***parent***)

pid_t fork();



Operacje na procesach - zakańczanie

Proces nadrzędny powinien zaczekać, aż proces podrzędny zakończy swoje działanie, aby pobrać kod wyjścia.

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);

Rodzaje procesów – zombi, sierota

Nie ma gwarancji który proces pierwszy zostanie urochomiony i zakończy się. Pojawia się zjawisko wyścigu (ang. ***race condition***).

Jeżeli proces-dziecko zakończył swoje działanie, a którego zamknięcie nie zostało obsłużone przez procesa-rodzica, wówczas ten proces-rodzic staje się procesem **zombi** (ang. ***zombie***).

Jeżeli proces-rodzic zakończy działanie przed procesem-dziecko, wówczas proces-dziecko staje się procesem **sierotą** (ang. ***orphan***).

Proces-rodzic jest „adoptowany” przez proces ***init*** albo przez proces powłoki (zmienia się jego ***ppid***).

Rodzaje procesów – demon

Proces działający w tle niewymagajacy interakcji z użytkownikiem, np:

- ftpd, czyli demon FTP
- httpd, czyli demon HTTP
- inetd, czyli demon Inet

Procedura tworzenia demona:

- stwórz proces potomy - ***fork()***
- zakończ proces rodzica – proces dziecka staje się sierotą (ang. ***orphan***)
- zmień prawa do plików
- wywołaj ***setsid()*** aby proces miał nową sesję oraz grupę
- zmień bieżący katalog na ”/”
- zamknij stdin, stdout, stderr

Wątki

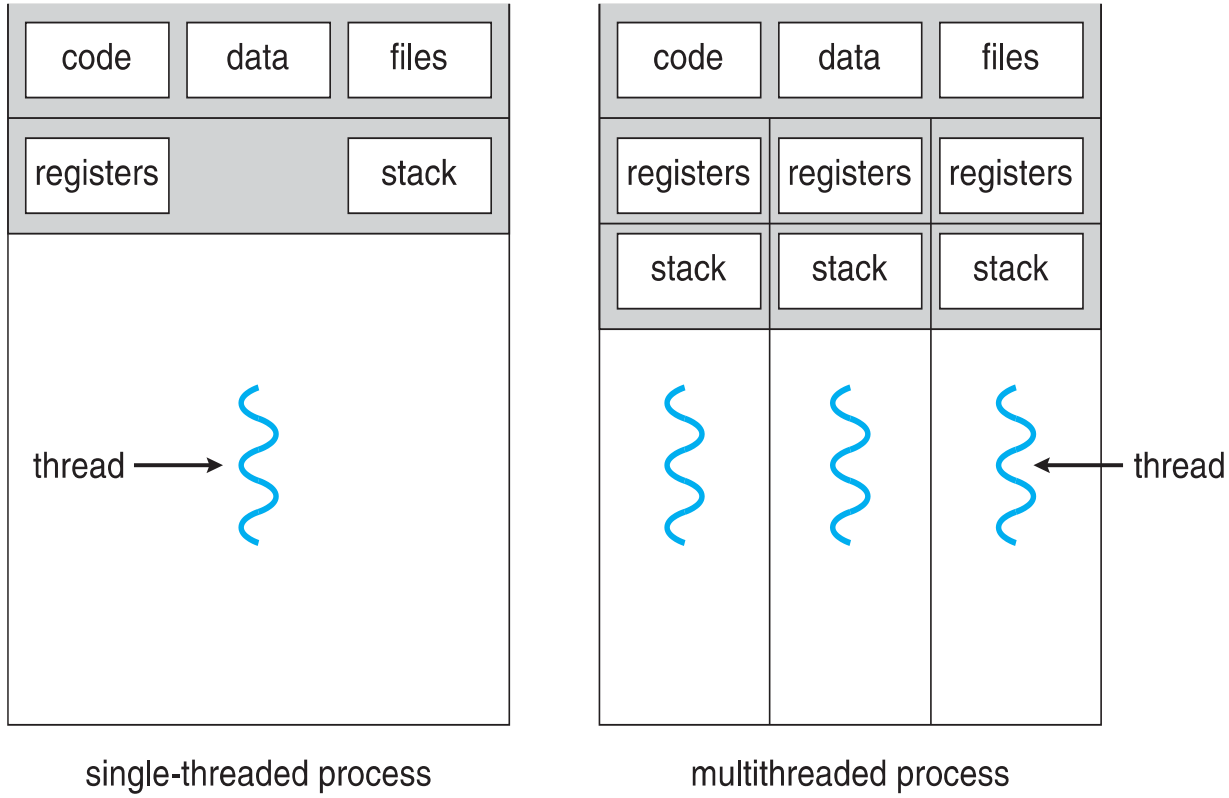
Pojęcie wątku (ang. *thread*)

Część programu wykonywana współbierznie w obrębie jednego procesu.

- zwane „procesami lekkimi” (ang. ***light weight processes***)
- w jednym procesie może istnieć wiele wątków
- **w ramach jednego procesu wątki współdzielą przestrzeń adresową oraz struktury systemowe**
- kompilacja z **-lpthread**

Wyświetlenie procesu wraz z wątkami

- ***ps -eLf | grep a.out***
- ***ps -m -l <pid>***



Zródła:
http://cs.nyu.edu/courses/fall12/CSCI-GA.2250-001/slides/

Wątki

- ile wątków dobrać ?
- jaką metodą synchronizacji wybrać ?
- na jakim sprzęcie będzie dział nasz wielowątkowy program ?
- którą część kodu zrównoleglić ?
- ile czasu zajmie wykonania naszego programu?
- a co jeśli się pomylimy ?

Co chcemy naprawdę ??? => **WYNIK bez blokowania programu a nie wątki !!!**

Zródła:
https://nerdist.com/china-builds-worlds-most-powerful-computer-and-its-not-even-
http://www.escapistmagazine.com/forums/read/704.936260-Who-remembers-their-first-computer

Wątki są SLOW

"Plain Threads are GOTO of Today's Computing" (for parallel computing)

Starvation

Insufficient concurrent work to maintain high utilization of resources.

Latencies

Time-distance delay of remote resource access and services.

Overhaeads

Work for management of parallel actions and resource on critical path which are necessary in sequential variant.

Waiting

Delays due to lack of availability of oversubscribed shared resources.

Zródła:
https://www.youtube.com/watch?v=4OCUEgSNIAY&list=WL&t=0s&index=15

Wątki alternatywa

Alternatywa dla C++:

- parallel algorithms
- std::future
- std::promise
- std::async
- std::package_task

Czyli:

- w kodzie powiedzmy tylko którą cześć kodu neleży zrównoleglić
- pozwolmy zdecydować językowi (systemowi operacyjnemu) jak optymanie zrównoleglić kod w zależności od sprzętu
- pozbądźmy się potencjanych błędów z synchronizacją / komunikacją pomiędzy wątkami

To co chcemy to **wynik**, który dostajemy od razu, **bez blokowania programu** – bezpośrednie operacje na wątkach nie są potrzebne.

Wątki alternatywa

jednak zanim to poznamy...

Wątki – interfejs (POSIX)

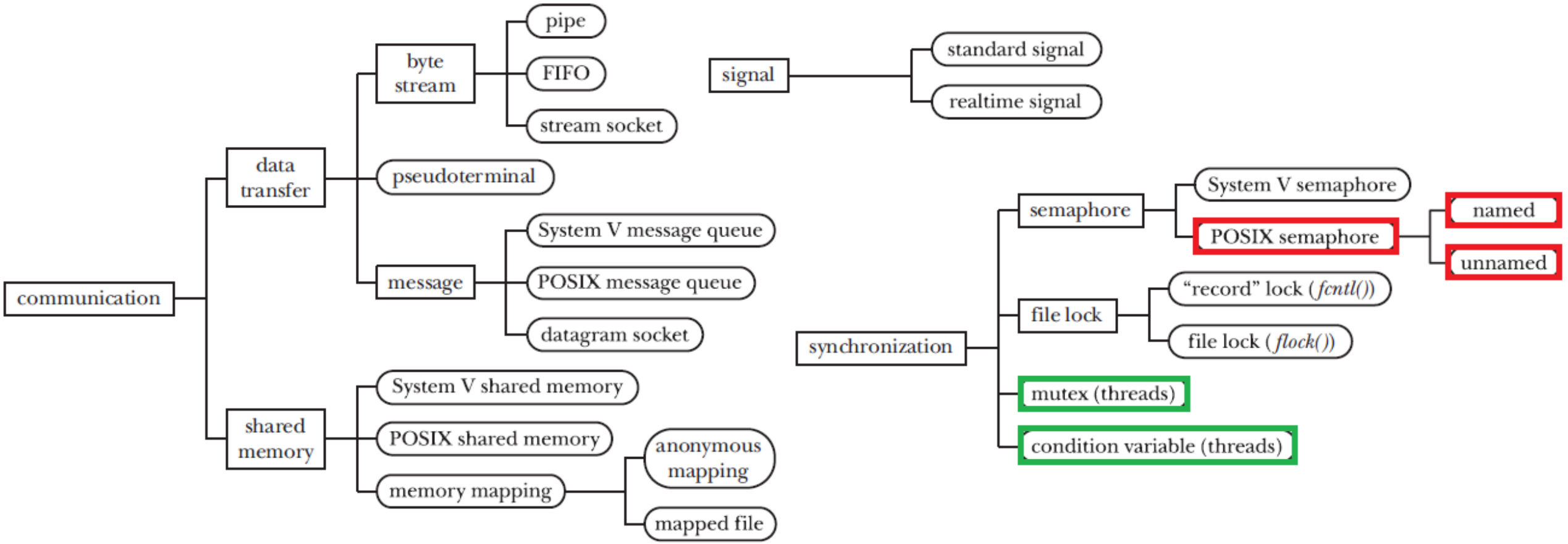
pthread_create(3)	tworzenie nowego wątku
pthread_attr_init(3)	inicjalizacja atrybutów wątku
pthread_attr_destroy(3)	usunięcie atrybutu wątku
pthread_attr_setdetachstate(3)	ustawienie wątku w tryb detached/joinable
pthread_self(3)	pobranie id wątku
pthread_detach(3)	ustawienie wątku jako ”detached”
pthread_join(3)	czekanie, aż wątek się zakończy
pthread_exit(3)	poprawne zakończenie wątku
pthread_cancel(3)	zakończenie jednego wątku przez inny wątek, ”cancelation request”
pthread_kill(3)	zakończenie jednego wątku przez inny wątek, sygnał

Wątki – sychronizacja i komunikacja

Podstawowe pojęcia

- **Komunikacja** (ang. ***Communication***) – mechanizmy służące do wymiany danch pomiędzy procesami/wątkami
- **Synchronizacja** (ang. ***Synchronization***) – mechanizmy służące do zgrania działania procesów/wątków
- **Sygnały** (ang. ***Signals***) – mechanizmy programowych przerwań mogące być wykorzystwane w roli komunikacji lub synchronizacji

Podstawowe pojęcia



Zródła:
„The Linux programming Interface", Michael Kerrisk

Podstawowe pojęcia

Sekcja krytyczna (ang. *Critical section*)
A section of code within a process that requires access to shared resources and that must not be executed while another process is in corresponding section of code.

Wyścig (ang. *Race condition*)
A situation in which multiple processes read and write a share data item and the final result depends on the relative timing of their execution.

Operacja atomowa (ang. *Atomic operation*)
A function or action implemented as sequence of one or more instruction that appears to be indivisible; that is, no other process can see and intermediate state or interrupt the operation. The sequence of instruction is guaranteed to execute as a group, to not execute at all, having no visible effect on system state.

Wzajemne wykluczenie (ang. *Mutual exclusion*)
The requirement that when once process is in a critical section that accesses shares resources no there process may be in a critical section that accesses any of those shared resources.

Zródła:
http://cs.nyu.edu/courses/fall12/CSCI-GA.2250-001/slides/Chapter05.pdf

Podstawowe pojęcia

Zagłodzenie (ang. *Starvation*)
A situation in which a runnable process is overlook indefinitely by the scheduler, although it is able to proceed, it is never chosen.

Zakleszczenie (ang. *Deadlock*)
A situation in which two or more process are unable to proceed because each is waiting for one of the others to do something.

Zakleszczenie (ang. *Livelock*)
A situation in which two or more process continuously change their state in response to change in the other process(es) without doing any useful work.

Zródła:
http://cs.nyu.edu/courses/fall12/CSCI-GA.2250-001/slides/Chapter05.pdf

Mutex (ang. *Mutex*)

- mechanizm służący do synchronizacji działa wątków poprzez atomowy dostęp do współdzielonego zasobu (ang. ***shared resource***)

- wzajemne wykluczanie (ang. ***mutual exclusion***)
- dwa stany:
  - zablokowany (ang. ***locked***) – ponowne blokowanie powoduje zablokowania działania wątku
  - odblokowany (ang. ***unlocked***)
- odblokować mutex może tylko ten sam wątek, co go blokował

Mutex - interfejs

pthread_mutex_init(3p)	inicjalizacja mutexu
pthread_mutex_lock(3p)	blokowanie mutexu
pthread_mutex_unlock(3p)	odblokowanie mutexu
pthread_mutex_destroy(3p)	zniszczenie mutexu

Zmienna warunku (ang. *Conditional variable*)

- mechanizm służący do synchronizacji działa wątków poprzez informowanie (ang. ***signaling***) innych wątków o stanie współdzielonego zasobu (ang. ***shared resource***)

- dopóki porządany stan wpółdzielonego zasobu nie zostanie osiągnięty wykonanie wątku może zostać zablokowane
- odblokowanie wątku może nastąpić przez inny wątek
- zawsze używana wraz z mutexem

Zmienna warunku - interfejs

pthread_cond_init(3p)	inicjalizacja zmiennej warunku
pthread_cond_signal(3p)	informowanie czekającego wątku, że określony warunek został spełniony
pthread_cond_broadcast(3p)	informowanie wszystkich czekających wątków, że określony warunek został osiągnięty
pthread_cond_wait(3p)	czekanie, że określony warunek zostanie spełniony
pthread_cond_destroy(3p)	zniszczenie zmiennej warunku

Semafor (ang. ***semaphore***)

- mechanizm służący do synchronizacji działa wątków/procesów

- posiadaja wartość, która nie może spaść poniżej 0
- próba zmiejszenia wartości poniżej 0, powoduje zabklokowanie działania
- inkrementacja/dekrementacja może nastąpić przez inny proces/wątek

- **nienazwane** (ang. ***unnamed***)
  - nieposiadają nazwy - muszą znajdować sie we wspólnej pamięci dla procesów/wątków
  - mogą być używane dla:
    - wątków
    - procesów powiązanych => *pshared != 0* w *sem_init()*
    - procesów niepowiązane => *pshared != 0* w *sem_init()*

- **nazwane** (ang. ***named***)
  - posiadają nazwe (***/dev/shm***) – nie muszą znajdować się we wspólnej pamięci
  - mogą być używane dla:
    - wątków
    - procesów powiązanych
    - procesów niepowiązane

Semafor - interfejs

sem_open(3)	otwiera/tworzy nazwany semafor
sem_init(3)	otwiera/tworzy nienazwany semafor
sem_post(3)	inkrementacja semafora
sem_wait(3)	dekrementacja semafora
sem_destroy(3)	zniszczenie nienazwanego semafora
sem_close(3)	zamknięcie nazwanego semafora
sem_unlink(3)	usunięcie nazwanego semafora

Uproszone zalecenie

Wątki:

- mutexy
- zmienne warunkowe

Procesy powiązane:

- semafory nienazwane

Procesy niepowiązane:

- semafory nazwane

Zadanie

Treść zadania

1. Zaimplementuj listę korzystając z dynamicznej alokacji pamięci.
2. Węzeł listy ma być zdefiniowany jako:

```
typedef struct List {
    int value;
    struct List* next;
} List;
```

3. Maksymalna ilość elementów na liście ma być konfigurowalna przez makro **MAX_ELEMS**.
4. Zaimplementuj metody dodawania/usuwania elementów na listę:

int push_back(int arg)

- agrument **arg** to wartość węzła **value**, który na być dodany na koniec listy. Jeżeli ilość elementów listy przekroczyła **MAX_ELEMS**, wówczas jest zwracana wartość **-1**

int pop_front()

- zwracana wartość to wartość pola **value** z początkowego węzła listy. Jeżeli lista jest pusta, wówczas jest zwracana wartość **-1**

5. Stwórz pulę wątków z czego:
   - co najmniej dwa wątki mają zapisywać dane (n-producentów)
   - co najmniej dwa wątki mają odczytywać dane (m-konsumentów)
6. Zidetyfikuj sekcje krytyczne w programie i odpowiednio je zabezpiecz.
7. Jeżeli lista jest pusta, wówczas czytające wątki mają być uśpione.
8. Jeżeli lista jest pełna, wówczas zapisujące wątki mają być uśpione.

schemat n-producentów, m-konsumentów, k-zasobów