Humboldt University

Computer Science Department
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Operating Systems Principles

Thread & Events

Lab - 0

- Make-Performance
  - Zusammenhang zwischen Projekt, #CPUs & #Jobs
  - Darstellen als Diagramm
  - Auswertung

- Tools
  - Make
  - Time
  - Tar/bz2/gzip
  - Matlab

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Events

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Events

- Programmieren mit Events unter Unix:
  - Signale
  - *select()*
  - *libboost (asio)*

- Event Queues (z.B in Simulatoren)
  - Jist
  - NS2

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Signale

signal

NAME

signal

SYNOPSIS

#include <signal.h>

void (*signal(int *sig*, void (**func*)(int)))(int);

DESCRIPTION

signal management

Determines how to handle the receiption of the signal number *sig* is to be subsequently handled

- SIG_DFL - default handling for that signal
- SIG_IGN - the signal shall be ignored
- Otherwise, *func* points to a function ("signal handler") to be called when that signal occurs

http://www.opengroup.org/onlinepubs/009695399/functions/signal.html

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

wait

NAME

wait, waitpid

SYNOPSIS

#include <sys/wait.h>

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

DESCRIPTION

Wait for a child process to stop or terminate

Obtain status information pertaining to one of the caller's child processes

If status information is available for two or more child processes, the order in which their status is reported is unspecified.

http://www.opengroup.org/onlinepubs/009695399/functions/wait.html

kill

NAME

kill

SYNOPSIS

#include <signal.h>

int kill(pid_t pid, int sig);

DESCRIPTION

Sends signal to a process or a group of processes specified by *pid*

Pids <= 0 used to send a signal to more than one process

Return value: 0 if successful ; -1 otherwise

Signals (examples):

SIGSTOP, SIGCONT

SIGTERM, SIGKILL

http://pubs.opengroup.org/onlinepubs/7908799/xsh/kill.html

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Select

Systems Architecture Group
http://sar.informatik.hu-berlin.de

select

NAME

select

SYNOPSIS

#include <sys/select.h>

int select(int *nfds*, fd_set *restrict *readfds*, fd_set *restrict *writefds*,
fd_set *restrict *errorfds*, struct timeval *restrict *timeout*);

int FD_ISSET(int *fd*, fd_set **fdset*);
void FD_SET(int *fd*, fd_set **fdset*);
void FD_ZERO(fd_set **fdset*);

DESCRIPTION

select() return if one or more of the file descriptor in the sets (*readfds*, *writefds*, and *errorfds)* are ready for reading, are ready for writing, or have an exceptional condition pending or due to a timeout

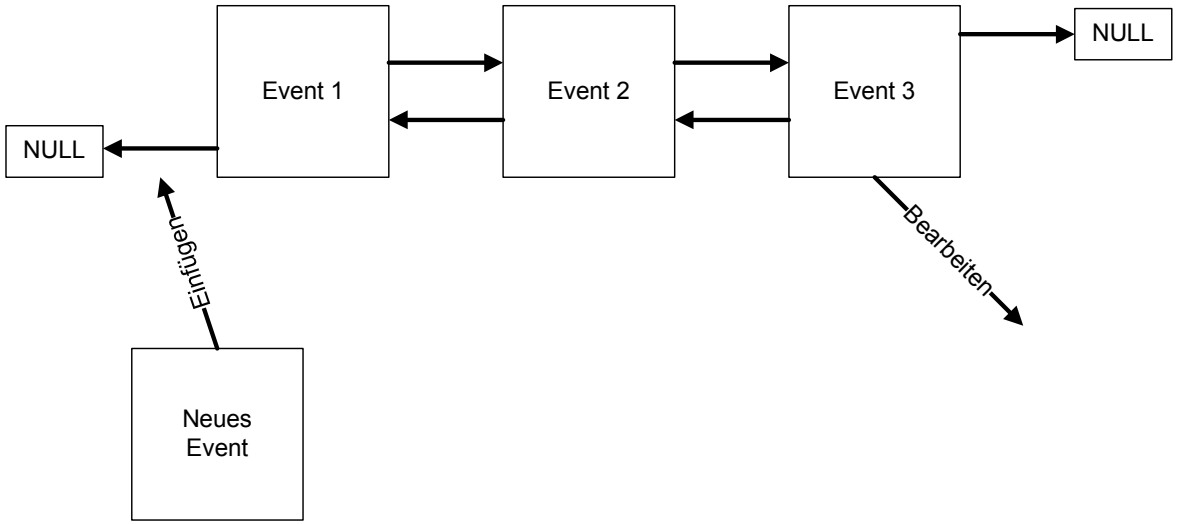
Eventqueue

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Eventqueue

- Event
  - Priorität
  - *Handler*
  - *Zeit*
  - *…*

- Event Queue
  - Verkettete Liste
  - Events:
    - Löschen
    - Einfügen
  - Scheduler arbeitet Events ab



Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Threads

POSIX-Threads (Pthreads)

Standard IEEE P1003.1c

In vielen Systemen realisiert

Konzepte in anderen Thread-Realisierungen meist ähnlich

POSIX-konforme Realisierungen unter Linux

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Die Pthread-Schnittstelle

Thread-Erzeugung

Programmiermodell

- Bei Start eines Prozesses existiert genau ein Thread
- Dieser erzeugt ggf. weitere Threads und wartet auf deren Ende
- Terminierung des Prozesses bei Terminierung des Master-Threads

Funktion zur Thread-Erzeugung

```
int pthread_create(
        pthread_t *new_thrd_ID,
        const pthread_attr_t *attr,
        void *(*start_func)(void *),
        void *arg)
```


Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Die Pthread-Schnittstelle

Thread-Attribute

Stackgröße

Scheduling-Schema und Priorität

detachstate: Verhalten bei Beendigung desThreads

Bei **detached thread** erfolgt die Freigabe der Ressourcen sofort bei Beendigung, ansonsten erst nach Abschlußsynchronisation

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Die Pthread-Schnittstelle

Thread-Verwaltung

pthread_self

Liefert eigene Thread-ID

pthread_exit

Eigene Terminierung

pthread_cancel

Terminiert einen Thread

Terminierung nur an bestimmten Punkten

Vor Terminierung **cleanup handler** aufrufen

Die Pthread-Schnittstelle

Thread-Verwaltung...

pthread_join

Wartet auf Terminierung des spezifizierten Threads (Abschlußsynchronisation)

pthread_sigmask

Setzt Signalmaske (jeder Thread hat eigene Maske)

pthread_kill

Sendet Signal an anderen Thread innerhalb des Prozesses

Von außen nur Signal an irgendeinen Thread möglich

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Die Pthread-Schnittstelle

Mutex-Operation (wechselseitiger Ausschluß)

pthread_mutex_init

Initialisiert Sperrvariable (mutex)

pthread_mutex_destroy

pthread_mutex_lock

Blockiert Thread, bis Sperre frei ist und belegt dann die Sperre

pthread_mutex_trylock

Belegt Sperre, falls möglich / kein Blockieren

pthread_mutex_unlock

Anmerkungen

Bei Terminierung eines Threads werden Sperren nicht automatisch freigegeben

Die Pthread-Schnittstelle

- Bedingungsvariablen
- Zur Signalisierung von Bedingungen zwischen Threads
- Erlauben Realisierung von Monitoren (strukturierte Form des wechselseitigen Ausschluß nach Hoare)
  - Extern sichtbare Funktionen eines Moduls stehen unter wechselseitigem Ausschluß
  - Aufrufer braucht sich damit nicht um Synchronisation kümmern

Backup

Organisatorisches

Datum	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
21.10. - 25.10.			Übung 0 Erste Aufgabe (Beginn)		
28.10. - 01.11.			Übung 1		
04.11. – 08.11.			Übung 2		Zweite Aufgabe (Beginn)
11.11. – 15.11.	Erste Aufgabe (Abgabe)		Übung 3		
18.11. – 22.11.			Übung 4		
25.11. – 29.11.			Übung 5		Dritte Aufgabe (Beginn)
02.12. – 06.12.	Zweite Aufgabe (Abgabe)		Übung 6		
09.12. – 13.12.			Übung 7		
16.12. – 20.12.			Übung 8		Vierte Aufgabe (Beginn)
23.12. – 27.12.					
30.12. – 03.01.					
06.01. – 10.01.	Dritte Aufgabe (Abgabe)		Übung 9		
13.01. – 17.01.			Übung 10		
20.01. – 24.01.			Übung 11		Fünfte Aufgabe (Beginn)
27.01. – 31.01.	Vierte Aufgabe (Abgabe)		Übung 12		
03.02. – 07.02.			Übung 13		
10.02. – 14.02.	Fünfte Aufgabe (Abgabe)		Übung 14		

Tools

- time

Systems Architecture Group
http://sar.informatik.hu-berlin.de