bash-Kommandos

exit/<Strg>D: beenden der Shell
expr : Arithmetische Ausdrücke, bei Vergleichen 0 = false
ls: Verzeichnis ausgeben
wc: worte zählen
echo: ausgabe
cd
cat
man /–help
read var1 var2 ...: lesen von eingabe in variable
-gt größer als -lt kleiner

Ströme

stdin: 0
stdout: 1
stderr: 2

Umleitung

> Datei überschreiben
» an Datei anhängen
< aus Datei lesen (in stdin)
| Pipe
& Prozess im Hintergrund starten
cd; ls Sequenz

Variablen

Defintion: var=12
Ausgabe: echo $var
alle Ausgeben: set

Parameter

$# Anzahl der Parameter
$* Alle Parameter (zusammengefasst)
$- übergebene Schalter (z.B. -a)
$@ Alle Parameter (einzeln)
$? Wert des letzten ausgeführten Kommandos
$_ letztes Argument des letzten Kommandos
$$ PID dieser Shell
$PID des letzten Hintergrundkommandos

Kontrollstrukturen

if

if Bedingung # if ["$v1- "$v2"]
then
[elif Bedingung
then]
[else]
fi
wahr: 0, falsch !=0

Mehrfachauswahl

case $var in
1) echo wert = 1
c) echo wert = c
*) echo Ungueltig
esac

for

for x in $Liste # for F in ‘find -name bla*‘
do
done

Zählschleife

for ((i = 1; i <= $max; i++))

while

while

do
done

until

until
do
done

Unterbrechungen

Polling

Busy waiting abfrage des Zustandes

Interrupts

Unterbrechen der Aktuellen Routine, ausführen der Interrupt Service routine, Auslöser Hardware oder Software(Trap)
Interrupt-Service-Routine: Kernel-Code der auf interrupt reagiert, wird durch Index in Interrupt-Vector-Table aufgerufen

Systemcall/Trap

Software-Interrupt zur Kommunikation mit BS, z.B. fork(), open(), close()...

Prozess

Prozesskontext: Zustandsinformation zum Prozess (Stack, Register,...) => Process-Control-Block: Programmzähler, Prozesszustand, Priorität ,Verbrauchte Prozessorzeit seit dem Start des Prozesses, Prozessnummer (PID), Elternprozess (PID), Zugeordnete Betriebsmittel z.B. Dateien
zustände: bereit, aktiv, beendet, blockiert
fork() klont Prozess, return 0 für kind, return kind-PID für Eltern

Scheduling

Zeitscheibe: zuteilung von zeit-Quanten an Prozesse, wechseln nach ablauf/Blockieren
Ziele: Fairness, Effizienz, Antwortzeit, Verweilzeit (Durchlaufzeit), Durchsatz
Non-Preemptive Scheduling vs Preemptive Scheduling (Prozess kann unterbrochen werden)

Strategien

- First Come First Served (FCFS): Der Reihe nach
- Shortest Job First (SJF); Theoretisch Optimal, kürzester Gewinnt
- Shortest Remaining Time Next (SRTN): kürzeste Restlaufzeit gewinnt, nicht preemtiv
- Round-Robin-Scheduling (RR) = Rundlauf-Verfahren: Der Reihe nach
- Priority Scheduling (PS) statisch/dynamisch: höchste Priorität gewinnt
- Shortest Remaining Time First (SRTF): SRTN preemtiv
- Lottery Scheduling: Zufällige Vergabe von CPU-Zeit

Echtzeit-Betriebssystem: garantierte zeiten; Tasks in endlosschleife
Parameter: Computation time C $\leq$ Deadline D $\leq$ Period T
=> Wiederholung nach kleinstem gemeinsamen Vielfachen der Perioden => Major Cycle / Hyperperiode
Rate Monotonic Scheduling (RMS): kürzeste Periode gewinnt
Earliest-Deadline First (EDF): nächste Deadline zuerst

Synchronisation

Race Conditions: gemeinsam genutzte Betriebsmittel, ergebnis abhänging von ausführungsreihenfolge
Kritische Abschnitte: logisch ununterbrechbare Code-bereiche; synchronisation zum gegenseitigen Ausschluss
Kriterien von Dijkstra: -Keine zwei Prozesse dürfen gleichzeitig in einem kritischen Abschnitt sein (mutual exclusion) - Keine Annahmen über die Abarbeitungsgeschwindigkeit und die Anzahl der Prozesse bzw. Prozessoren - Kein Prozess außerhalb eines kritischen Abschnitts darf einen anderen Prozess blockieren - kein ewiges Warten (fairness condition)
Methoden:

- busy waiting/spinlock: testen einer Variablen bis zutritt erlaubt
- interrupts maskieren: nur bei Monoprozessoren, sehr ungünstig
- Hardwareunterstützung durch atomare Befehle

- Semaphore/Mutex

```
Semaphore x = new Semaphore();
x.Down(); // kritischer Abschnitt besetzt?
c=counter.read(); // kritischer Abschnitt
c++;
counter.write(c);
x.Up();  // Verlassen des kritischen Abschnittes
```

Erzeuger-Verbraucher

```
Erzeuger                          Verbraucher
While (true) {                    While (true) {
produce(item);                    Down(belegt);
Down(frei);                       Down(mutex);
Down(mutex);                      getFromBuffer(item);
putInBuffer(item);                Up(mutex);
Up(mutex);                        Up(frei);
Up(belegt);                       }
}
```

Monitor: eine Menge von Prozeduren und Datenstrukturen, die als Betriebsmittel betrachtet werden und mehreren Prozessen zugänglich sind, aber nur von einem Prozess/Thread zu einer Zeit benutzt werden können

Deadlock

Darstellung: Belegungsgraph; Prozess -> Ressource
Bedingungen: Mutual exclusion: Ressourcensharing nicht möglich (DVD-Brenner) Hold-and-wait: Prozesse belegen Ressourcen und wollen weitere No preemption: Entzug nicht möglich Circular waiting: gegenseitiges Warten
Strategien: Ignorieren (wenn selten)
Erkennen und beheben (Erkennen anhand Belegungsgraph) : Unterbrechung, Rollback Prozessabbruch Transaktionsabbruch
Dynamisches Verhindern: notwendig Vorwissen über Bedarf z.B. Bankiers-Algorithmus: prüfen, ob es eine Zuteilungsreihefolge gibt, bei der der Bedarf erfüllt werden kann
Vermeiden: Mutual exclusion: z.B. virtualisieren mit Spooling Hold-and-wait: anfordern aller benötigten Ressourcen auf einen Schlag, oder freigabe alter Ressourcen bevor weitere Angefordert werden No preemption: Entzug nicht möglich Circular waiting: nummerieren der Ressourcen, anforderung nur in aufsteigender Reihenfolge
Echtzeitsysteme: Priority Ceiling Protocol Ressource hat Ceiling Priorität = maximale Priorität der Tasks, die sie verwenden werden. Der sie nutzende Task hat während der Nutzung diese Priorität

Kommunikation

Nachrichten: verbindungsorientiert vs verbindungslos
Speicher: gemeinsamer Adressraum (Threads), Shared Memory, Datei (Prozess)
Synchron(Blockierend) vs Asynchron

Interprozesskommunikation

- Pipes und FIFOs (Named Pipes) als Nachrichtenkanal - Nachrichtenwarteschlangen (Message Queues) - Gemeinsam genutzter Speicher (Shared Memory) - Sockets (Ip-Loopback)

Pipes

Unidirektional, bidirektional über mehrere Pipes; Standardausgabe zu Standardeingabe

```
int fds[2] / Filedescriptoren für Pipe
pipe(fds);
if (fork() == 0) {
// 1. Kindprozess, Standardausgabe auf Pipe-Schreibseite (Pipe-Eingang) le
dup2(fds[1], 1); // 1 = Standardausgabe
close(fds[0]);
write (1, text, strlen(text)+1);
}
else{
if (fork() == 0) {
// 2. Kindprozess, Pipe-Leseseite (Pipe-Ausgang) auf
// Standardeingabe umlenken und Pipe-Schreibseite
// (Pipe-Eingang) schließen
dup2(fds[0], 0); // 0 = standardeingabe
close(fds[1]);
while (count = read(0, buffer, 4))
```

```
{
// Pipe in einer Schleife auslesen
prozess Pipe
buffer[count] = 0; // String terminieren
printf(“%s“, buffer) // und ausgeben
}
else {
// Im Vaterprozess: Pipe an beiden Seiten schließen und
// auf das Beenden der Kindprozesse warten
close(fds[0]);
close[fds[1]);
wait(&status);
wait(&status);
}
exit(0);
}
}
```