Konzepte von Betriebssystem-Komponenten
Ausnahme- / Unterbrechungsbehandlung

Sommersemester 2005

Uni Erlangen
Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebsysteme

Tarek Gasmi
Tarek.Gasmi@informatik.stud.uni-erlangen.de

❖ Gliederung:

❖ Einführung:

Während einer Programmausführung können Ereignisse eintreten, die eine spezielle Reaktion durch das System erfordern. Solche Ereignisse werden außerhalb des gerade laufenden Prozesses behandelt.
Diese Behandlung ist zwingend und grundsätzlich prozessorabhängig: egal ob der Prozessor abstrakt (virtuelle Maschine) oder physikalisch (reale Maschine) ist.
Die Unterbrechungen unterscheiden sich in:

- ***Quelle***
- ***Synchronität***
- ***Vorhersagbarkeit***
- ***Reproduzierbarkeit***

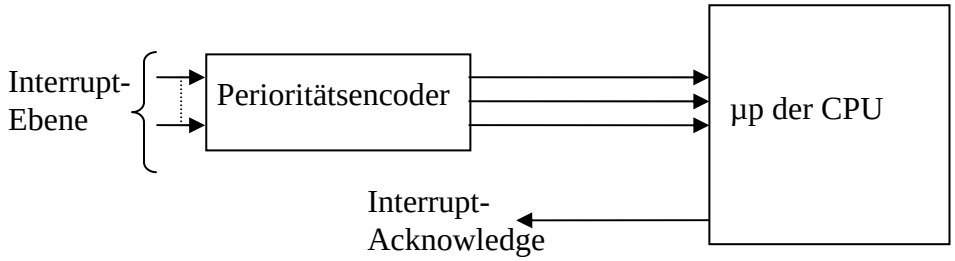
Man **unterscheidet** zwischen ***Traps*** und ***Interrupts***.

- Trap: ist Ausnahme, die durch das Anwenderprogramm verursacht wird.
- Interrupt: Wunsch externer Systemkomponenten, die Aufmerksamkeit des Prozessors zu erhalten.

❖ Gemeinsamkeiten:

o Hardware-Aktivitäten:

- unabhängig von der laufenden Busoperation meldet eine Komponente über eine Interruptleitung ihren Unterbrechungswunsch an.
- Der Prioritätsencoder leitet immer nur den Interrupt mit der jeweils höchsten Priorität zur CPU weiter.
- Die CPU beendet normal den aktuellen Befehl und rettet sowohl ihr Statusregister wie auch den Befehlszähler auf der Stack.



- Zu jedem Interrupt soll nun das entsprechende Unterprogramm, die so genannte Interrupt-Service-Routine, aufgerufen werden. Der Anfang eines Unterprogramms, ist in der Vektor-Tabelle (fester Bereich in Hauptspeicher) abgelegt.

Adresse	Periorität	Interrupt-Quelle
20	15 (höchste)	Timer
24	14	Tastatur
28	-	(interrupt om Interrupt-Controller2)
02C	5	Serielle Schnittstelle 2
30	4	Serielle Schnittstelle 1
34	3	Parallele Schnittstelle 2 (früher: harddisck)
38	2	Floppy Disk
03C	1 (niedrigste)	Parallele Schnittstelle 1
1C0	13	Echtzeituhr
1C4	12	I/O-Kanal
1C8	11	I/O-Kanal
1CC	10	I/O-Kanal
1D0	9	I/O-Kanal
1D4	8	Co-Prozessor
1D8	7	Harddisk
1DC	6	I/O-Kanal

Für jede Mikroprozessorfamilie gibt es einen einheitlichen definierten Speicherbereich, in dem die Vektor-Tabelle festgelegt ist. In unserem Beispiel: Timer hat die höchste Priorität (15) und seine Interrupt-Service-Routine ist in der Adresse 20. Die Tastatur belegt die zweite höchste Priorität und seine Interrupt-Service-Routine ist in der Adresse 24. Und die niedrigste Priorität (1) ist für Parallel Schnittstelle 1 und seine Interrupt-Service-Routine ist in der Adresse 03C.

- Software-Aktivitäten:
  - Die Interrupt Service Routine stellt ein Unterprogramm dar. Also müssen zu nächst (wie bei jedem Unterprogramm) alle Register, die im Unterprogramm zum Einsatz kommen, auf den Stack gerettet werden(1).
  - In der Interrupt Service Routine wird der Interrupt bearbeitet.
  - Vor dem Rücksprung zum unterbrochenen Programm lädt die CPU aus dem Stack die Register, die in (1) gerettet wurden.

❖ *Traps:*

Traps werden durch den geraden laufenden Prozess verursacht, befinden sich also auch in einem zeitlichen Zusammenhang („synchron“) mit dem Prozess.
Ein in die Falle gelaufene („getrappts“) Prozess, der sich unverändert wiederholt, der jedes Mal mit den selben Eingabedaten versorgt und auf dem selben Prozessor zur Ausführung gebracht wird, wird auch immer wieder an der selben Stellen in die selben Fallen laufen, d.h. den selben Trap verursachen.

Beispiele für Traps sind:

- *Überlauf bei einer arithmetischen Operation*
- *Eine Division durch 0*
- *Unbekannter Befehl*
- *Systemaufruf*
- *Ein Fehler bei einem Speicherzugriff*

☞ Ohne Behebung der Ausnahmebedingung ist eine Trap-Vermeidung unmöglich.

```
int main(int argc, char **argv){
        int a,b;
        scanf(a,b);
        printf("a/b : %f \n",a/b);
        return 0;
}
```

Wenn man a=5 und b=0 eingibt, tritt ein Trap ein. Ohne Änderung im Code tritt immer der gleiche Trap ein.
Der Programmierer kann überprüfen, ob b null ist, und eine Fehlermeldung ausgeben.

```
int main(int argc, char **argv){
        int a,b;
        scanf(a,b);
        if( b == 0 ){
                fprintf(stderr,"dividieren  durch null ist nicht möglich");
                return 1;
        }
        printf("a/b : %f \n",a/b);
        return 0;
}
```

Traps werden nicht durch das Benutzerprogramm, sondern durch die Hardware oder das Mikroprogramm des Systems entdeckt. Das hat zwei Vorteile:

- Der Programmieraufwand und die Ausführungszeit werden eingespart, die bei einer expliziten Abfrage von Fehlermöglichkeiten durch das Programm anfielen.
- Das System selbst reagiert auf das Ereignis. Das ist sicherer, als wenn man dem Benutzerprogramm die Behandlung kritischer Fehlerfälle überließe.

❖ *Interrupts :*

Interrupts sind externe und damit asynchrone Unterbrechungen. Sie sind Änderung in der Ablaufsteuerung, die nicht vom laufenden Prozess, sondern von etwas anderem (meist in Zusammenhang mit E/A-Geräten) verursacht werden.
Beim Interrupt hält der Prozessor den laufenden Prozess an und gibt er die Kontrolle an einen Interrupt-Handler ab, der eine bestimmte Aktion durchführt. Nach Beendigung gibt die Kontrolle wieder an den unterbrochenen Prozess zurück. Er muss den unterbrochenen Prozess in genau demselben Zustand fortsetzen, in dem er unterbrochen wurde. Das bedeutet, dass alle internen Register auf den Zustand, der vor dem Interrupt herrschte, zurückgesetzt werden müssen.

Beispiele für Interrupts sind:

- *Timer*
- *Eine Taste gedrückt*
- *E/A-Signale*

☞ Die Behandlung der Ausnahmesituation muss nebeneffektfrei verlaufen

Während einer Programmausführung kann ein Timer-Interrupt eintreten. Der Prozessor soll den Interrupt bearbeiten, dann springt er zum unterbrochenen Programm zurück. Es kann sein, dass während der Interrupt-Bearbeitung, ein Statusregister oder der Befehlszähler geändert wird. Das muss nicht sein.

❖ *Der Unterschied zwischen Interrupt und Trap:*

1. Ein ***Trap*** ist synchron, vorhersagbar, reproduzierbar und kein Interrupt.
2. Ein ***Interrupt*** ist asynchron, unvorhersagbar, nicht reproduzierbar und kein Trap.

Beispiele:

1. Wenn manche Geräte (Z.B. Zeitgeber) Interrupts in gleichen zyklischen Zeitabständen reproduzieren können, ist die Stelle der Unterbrechung (d.h. der unterbrochene Maschinenbefehl) nicht vorhersagbar. Interrupts sind jedoch vorhersagbar in dem Sinne, dass (je nach Systemkonfiguration) mit ihrem Auftreten zu rechnen ist.
2.

```
float dividieren ( int a , int b){
        return a/b;
}
```

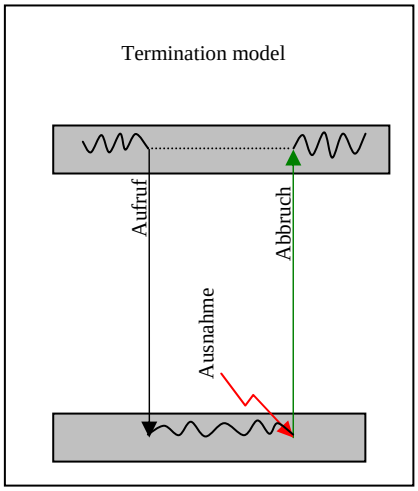
Eine Division durch 0 ist möglich.
Je nach CPU (bzw. ALU) führt dies zur Programmunterbrechung.
Ist diese Unterbrechung Trap oder Interrupt?

- diese Stelle ist **vorhersagbar**.
- Ohne Fehlerbeseitigung ist diese Unterbrechung somit **reproduzierbar**.
- Die Unterbrechung ist **synchron** zur Programmführung.
  - ***Die Unterbrechung ist ein Trap.***

❖ *Die Behandlung der Interrupts und Traps:*

Es sind zwei Konzepte zu unterscheiden:

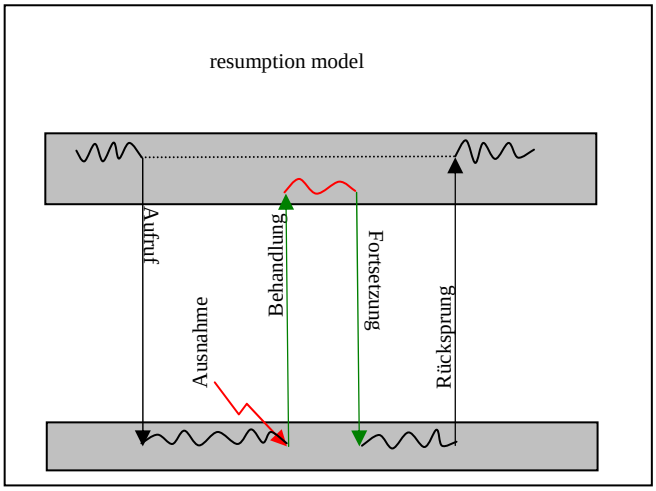
➢ Termination model :



Die Ausnahmesituation konnten nicht behandelt werden, es wird ein schwerwiegender Fehler angenommen, der zum Abbruch des unterbrochenen Programms führen muss. Ein Trap kann, aber ein Interrupt darf niemals nach diesem Modell behandelt werden.

Warum darf ein Interrupt niemals nach diesem Modell behandelt werden?
Ein Interrupt ist eine externe Unterbrechung und der Anwender will nicht, dass seinen ausgeführten Prozess von der externe Ausnahmesituation abhänge. (Z.B. Anwenderprogramm wird nach jeder Timer-Interrupt abgebrochen, somit kann der Anwender kein Programm ausführen lassen, dass sein Zeitaufwand länger als der Timer-Zeit ist).

➢ Resumption model:

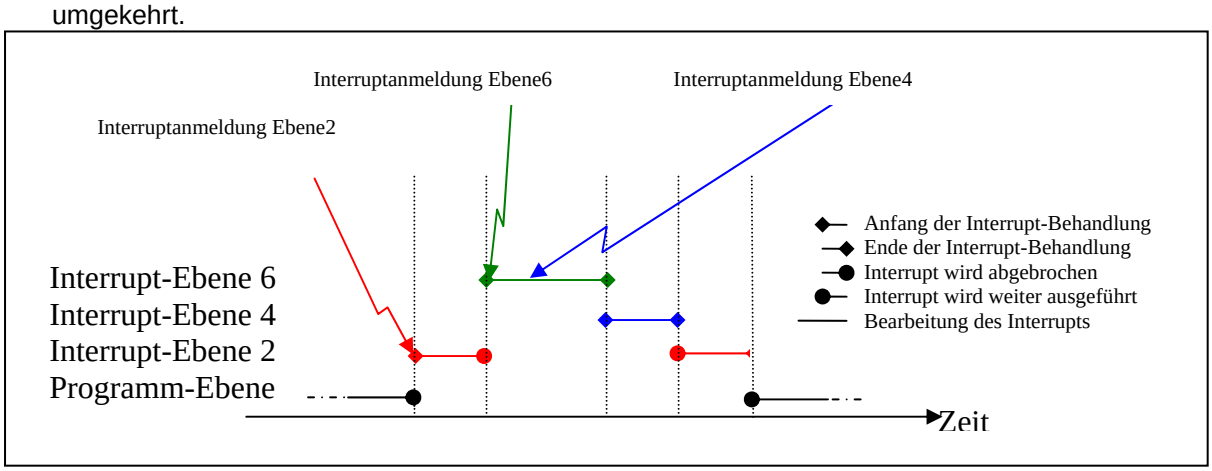


Die erfolgreiche Behandlung der Ausnahmesituation führt zur Wiederaufnahme der Ausführung des unterbrochenen Programms. Ein Trap kann aber ein Interrupt muss nach diesem Modell behandelt werden. Das unterbrochene Programm muss nach der Interrupt-Behandlung weiter ausgeführt werden.

❖ Mehrere Unterbrechungsebenen:

Es gibt die CPUs mit mehreren Unterbrechungsebenen. Bei solchen CPUs kann ein Interrupt einen anderen Interrupt mit niedriger Priorität unterbrechen. Deshalb vergleicht die CPU zuerst mittels der Interrupt-Maske in ihrem Status-Register, ob er neu gemeldete Interrupt eine höhere Ebene hat als ein eventuell gerade zu bearbeitender Interrupt. Der Programmierer kann die Interrupt-Maske setzen und damit Interrupts zeitweise maskieren, d.h. solche Interrupts unter einer bestimmten Ebene sperren. Interrupts mit niedrigerer Ebene müssen warten, bis sie wieder freigegeben werden.

- ➢ Ein Interrupt kann den Interrupt einer niedrigeren Ebene (Priorität) unterbrechen, aber nicht umgekehrt.



❖ Zusammenfassung:

Man kann die Unterbrechungen in zwei Kategorien einteilen Traps und Interrupts.

- ***Traps*** werden durch das gerade ausgeführte Programm verursacht.
- ***Interrupts*** sind externe Unterbrechungen.

Gemeinsamkeiten:

- Hardware-Aktivitäten
- Software-Aktivitäten

Der Unterschied zwischen Interrupt und Trap:

- Ein ***Trap*** ist synchron, vorhersagbar, reproduzierbar und kein Interrupt.
- Ein ***Interrupt*** ist Asynchron, unvorhersagbar, nicht reproduzierbar und kein Trap.

Die Behandlung der Interrupts und Traps:

Bei Behandlung der Interrupts und Traps gibt es zwei Konzepte:

- Termination model :

  Die Ausnahmesituation konnten nicht behandelt werden, wird ein schwerwiegender Fehler konstatiert, der zum Abbruch des unterbrochenen Programms führen muss. Ein Trap kann, aber ein Interrupt darf niemals nach diesem Modell behandelt werden.

- Resumption model:

  Die erfolgreiche Behandlung der Ausnahmesituation führt zur Wiederaufnahme der Ausführung des unterbrochenen Programms. Ein Trap kann, aber ein Interrupt muss nach diesem Modell behandelt werden.

Mehrere Unterbrechungsebenen :

Mehr als ein Interrupt können sich anmelden und nur Ein wird behandelt

- ➢ Ein Interrupt kann dem Interrupt einer niedrigeren Ebene (Priorität) unterbrechen, aber nicht umgekehrt.

❖ Literatur:

- o *Jürgen Nehmer and Peter Sturm, März 2001, **Systemsoftware: Grundlagen Moderner Betriebsystem**, Dpunkt.*
- o *Helmut Malz, 2004, **Rechnerarchitektur**, Broschiert-Vieueg-Verlagsgesellschaft.*
- o *Carsten Vogt, April 2001, **Betriebsysteme**, Spektrum Akademischer Verlag.*
- o *Andrew s. Tannenbaum James Goodman,15.Januar, **Computerarchitektur: Strukturen Konzepte Grundlagen**, Pearson Studium.*
- o *Wolfgang Schröder-Preikschat, S.S. 2004, **Skript der Softwaresysteme1**, URL: http://www4.informarik.uni-erlangen.de/Lehre/SS04/V_SOS1/Folien, 16-Juni-2004.*