

```

Sep 19 14:20:18 amd64 sshd(20494): Accepted rsa for esser from ::ffff:87.234.201.207 port 61557
Sep 19 14:20:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 01:00:01 amd64 /usr/sbin/cron[2978]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 20 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 02:00:01 amd64 /usr/sbin/cron[30103]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 20 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:46:44 amd64 sshd(6516): Accepted rsa for esser from ::ffff:87.234.201.207 port 62004
Sep 20 12:46:44 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 12:48:41 amd64 sshd(6609): Accepted rsa for esser from ::ffff:87.234.201.207 port 62105
Sep 20 12:54:44 amd64 sshd(6694): Accepted rsa for esser from ::ffff:87.234.201.207 port 62514
Sep 20 15:27:36 amd64 sshd(9077): Accepted rsa for esser from ::ffff:87.234.201.207 port 64242
Sep 20 15:27:36 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:37:11 amd64 sshd(10102): Accepted rsa for esser from ::ffff:87.234.201.207 port 63375
Sep 20 16:37:11 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 20 16:38:10 amd64 sshd(10201): Accepted rsa for esser from ::ffff:87.234.201.207 port 63546
Sep 21 01:00:01 amd64 /usr/sbin/cron[17055]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 21 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 04:00:01 amd64 /usr/sbin/cron[17878]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 21 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:43:26 amd64 sshd(31088): Accepted rsa for esser from ::ffff:87.234.201.207 port 63397
Sep 21 17:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 17:53:39 amd64 sshd(31269): Accepted rsa for esser from ::ffff:87.234.201.207 port 64391
Sep 21 18:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 21 19:43:26 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 01:00:01 amd64 /usr/sbin/cron[14674]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 22 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 02:00:01 amd64 /usr/sbin/cron[16991]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 22 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 22 20:23:21 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 01:00:01 amd64 /usr/sbin/cron[124739]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 02:00:01 amd64 /usr/sbin/cron[12473]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 23 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:05 amd64 sshd(6554): Accepted publickey for esser from ::ffff:192.168.1.5 port 59771 ssh2
Sep 23 18:04:05 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 23 18:04:34 amd64 sshd(6606): Accepted rsa for esser from ::ffff:87.234.201.207 port 62093
Sep 24 01:00:01 amd64 /usr/sbin/cron[12436]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 24 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 02:00:01 amd64 /usr/sbin/cron[13253]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 24 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 11:15:48 amd64 sshd(20999): Accepted rsa for esser from ::ffff:87.234.201.207 port 64456
Sep 24 11:15:48 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 13:49:08 amd64 sshd(23197): Accepted rsa for esser from ::ffff:87.234.201.207 port 61330
Sep 24 13:49:08 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_midi_event: unsupported module, tainting kernel.
Sep 24 15:42:07 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 24 15:42:07 amd64 kernel: snd_seq_oss: unsupported module, tainting kernel.
Sep 24 20:25:39 amd64 sshd(29399): Accepted rsa for esser from ::ffff:87.234.201.207 port 62566
Sep 24 20:25:39 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 01:00:01 amd64 /usr/sbin/cron[1662]: (root) CMD (/sbin/evlogmgr -c "severity=DEBUG")
Sep 25 01:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 02:00:01 amd64 /usr/sbin/cron[1484]: (root) CMD (/sbin/evlogmgr -c "age > *30d")
Sep 25 02:00:01 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:25 amd64 sshd(8889): Accepted rsa for esser from ::ffff:87.234.201.207 port 64183
Sep 25 10:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 10:59:47 amd64 sshd(8921): Accepted rsa for esser from ::ffff:87.234.201.207 port 64253
Sep 25 11:30:01 amd64 sshd(9372): Accepted rsa for esser from ::ffff:87.234.201.207 port 62029
Sep 25 11:59:25 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:05:37 amd64 sshd(11554): Accepted rsa for esser from ::ffff:87.234.201.207 port 62822
Sep 25 14:05:37 amd64 syslog-ng[7653]: STATS: dropped 0
Sep 25 14:06:10 amd64 sshd(11586): Accepted rsa for esser from ::ffff:87.234.201.207 port 62951
Sep 25 14:07:17 amd64 sshd(11608): Accepted rsa for esser from ::ffff:87.234.201.207 port 63392
Sep 25 14:08:33 amd64 sshd(11630): Accepted rsa for esser from ::ffff:87.234.201.207 port 63709
Sep 25 15:25:33 amd64 sshd(12930): Accepted rsa for esser from ::ffff:87.234.201.207 port 62778

```

7 Deadlocks (2)

/home/esser/Daten/Dozent/Folien/bs-esser-22.cdp

Deadlock-Vermeidung (2)

Sichere vs. unsichere Zustände

- Ein Zustand heißt **sicher**, wenn es eine Ausführreihenfolge der Prozesse gibt, die auch dann keinen Deadlock verursacht, wenn alle Prozesse sofort ihre maximalen Ressourcenforderungen stellen.
- Ein Zustand heißt **unsicher**, wenn er nicht sicher ist.
- Unsicher bedeutet nicht zwangsläufig Deadlock!

Deadlock-Vermeidung (1)

Deadlock Avoidance (Vermeidung)

- **Idee:** BS erfüllt Ressourcenanforderung nur dann, wenn dadurch auf keinen Fall ein Deadlock entstehen kann
- Das funktioniert nur, wenn man die **Maximalforderungen aller Prozesse** kennt
 - Prozesse registrieren **beim Start** für alle denkbaren Ressourcen ihren Maximalbedarf
 - für die Praxis i. d. R. irrelevant
 - nur in wenigen Spezialfällen nützlich

Deadlock-Vermeidung (3)

Banker-Algorithmus (1)

- Idee: Liquidität im Kreditgeschäft
 - Kunden haben eine Kreditlinie (maximaler Kreditbetrag)
 - Kunden können ihren Kredit in Teilbeträgen in Anspruch nehmen, bis die Kreditlinie ausgeschöpft ist – dann zahlen sie den kompletten Kreditbetrag zurück
 - Prüfe bei Kreditanforderung, ob diese die Bank in einem „sicheren“ Zustand lässt, was die Liquidität angeht – wird der Zustand unsicher, lehnt die Bank die Auszahlung ab

Deadlock-Vermeidung (4)

Banker-Algorithmus (2) – Beispiel

Bank: 1200 €,
900 € verliehen, 300 € Cash

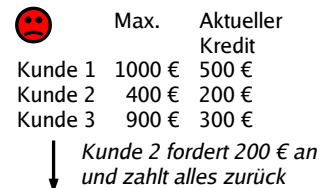
	Max.	Aktueller Kredit
Kunde 1	1000 €	500 €
Kunde 2	400 €	200 €
Kunde 3	900 €	200 €



Übergang sicher → unsicher
nicht zulassen!

Bank: 1200 €,
1000 € verliehen, 200 € Cash

	Max.	Aktueller Kredit
Kunde 1	1000 €	500 €
Kunde 2	400 €	200 €
Kunde 3	900 €	300 €



Bank: 400 € Cash
Kunde 1 1000 € 500 €
Kunde 2 400 € 0 €
Kunde 3 900 € 300 €



Deadlock-Vermeidung (6)

Banker-Algorithmus (4)

Feststellen, ob ein
Zustand sicher ist

=

Annehmen, dass alle Prozesse
sofort ihre Maximalforderungen
stellen, und dies auf Deadlocks
überprüfen

Deadlock-Vermeidung (5)

Banker-Algorithmus (3)

- Datenstrukturen wie bei Deadlock-Erkennung:
 - n Prozesse $P_1 \dots P_n$, m Ressourcentypen $R_1 \dots R_m$ mit je E_i Ressourcen-Instanzen ($i=1, \dots, m$)
→ **Ressourcenvektor** $E = (E_1 \ E_2 \ \dots \ E_m)$
 - **Ressourcenrestvektor** A (wie viele sind noch frei?)
 - **Belegungsmatrix** C
 C_{ij} = Anzahl Ressourcen vom Typ j , die Prozess i belegt
 - **Maximalbelegung** Max :
 Max_{ij} = max. Bedarf, den Prozess i an Ressource j hat
 - **Maximale zukünftige Anforderungen**: $R = Max - C$,
 R_{ij} = Anzahl Ressourcen vom Typ j , die Prozess i noch maximal anfordern kann

Deadlock-Verhinderung (1)

- Deadlock-Verhinderung (prevention):
Vorbeugendes Verhindern
 - mache mindestens eine der vier Deadlock-Bedingungen unerfüllbar
 1. gegenseitiger Ausschluss
 2. Hold and Wait
 3. Ununterbrechbarkeit der Ressourcen
 4. Zyklisches Warten
 - dann sind keine Deadlocks mehr möglich (denn die vier Bedingungen sind notwendig)

Deadlock-Verhinderung (2)

1. Gegenseitiger Ausschluss

- Ressourcen nur dann exklusiv Prozessen zuteilen, wenn es keine Alternative dazu gibt
- Beispiel: Statt mehrerer konkurrierender Prozesse, die einen gemeinsamen Drucker verwenden wollen, eine Drucker-Spooler einführen
 - keine Konflikte mehr bei Zugriff auf Drucker (Spooler-Prozess ist der einzige, der direkten Zugriff erhalten kann)
 - aber: Problem evtl. nur verschoben (Größe des Spool-Bereichs bei vielen Druckjobs begrenzt?)

Deadlock-Verhinderung (4)

3. Ununterbrechbarkeit der Ressourcen

- Ressourcen entziehen?
- siehe Deadlock-Behebung (Abbruch / Rücksetzen)

Deadlock-Verhinderung (3)

2. Hold and Wait

- Alle Prozesse müssen die benötigten Ressourcen gleich beim Prozessstart anfordern (und blockieren)
- hat verschiedene Nachteile:
 - Ressourcen-Bedarf entsteht oft dynamisch (ist also beim Start des Prozesses nicht bekannt)
 - verschlechtert Parallelität (Prozess hält Ressourcen über einen längeren Zeitraum)
- Datenbanksysteme: **Two Phase Locking**
 - Sperrphase: Alle Ressourcen erwerben (wenn das nicht klappt -> alle sofort wieder freigeben)
 - Zugriffsphase (anschließend Freigabe)

Deadlock-Verhinderung (5)

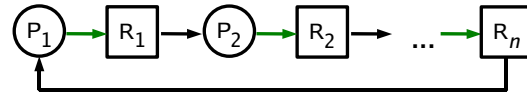
4. Zyklisches Warten (1)

- Ressourcen durchnummerieren
 - $ord: R = \{R_1, \dots, R_n\} \rightarrow \mathbb{N}$, $ord(R_i) \neq ord(R_j)$ für $i \neq j$
- Prozess darf Ressourcen nur in der durch ord vorgegebenen Reihenfolge anfordern
 - Wenn $ord(R) < ord(S)$, dann ist die Sequenz
 - lock (S);
 - lock (R);
 - ungültig
- Das macht Deadlocks unmöglich

Deadlock-Verhinderung (6)

4. Zyklisches Warten (2)

- Annahme: Es gibt einen Zykel



Für jedes i gilt: $ord(R_i) < ord(R_{i+1})$ und wegen des Zyklus auch $ord(R_n) < ord(R_1)$,
daraus folgt $ord(R_1) < ord(R_1)$: Widerspruch

- Problem: Gibt es eine feste Reihenfolge der Ressourcenbelegung, die für alle Prozesse geeignet ist?
- reduziert Parallelität (Ressourcen zu früh belegt)