3 Übungsaufgabe #3: Nachrichtenaustausch und erweitertes Paging

Ziel dieser Aufgabe ist es, den isolierten Prozessen effiziente Primitiven zum Nachrichtenaustausch (*message passing*) anzubieten, sowie einen Systemaufruf zur Prozesserzeugung, ähnlich dem Unix-`fork()`, bereitzustellen.

3.1 Systemaufrufe

Folgende Systemaufrufe sollen implementiert werden:

`int getpid()`
: liefert die Prozess-Identifikationsnummer des aufrufenden Prozesses. Diese Nummer soll im laufenden System eindeutig sein.

`void send(int id, void *sbuffer, size_t ssize, void *rbuffer, size_t rsize)`
: sendet die Nachricht in `sbuffer` mit der Länge `ssize` synchron und blockierend an den Prozess mit der Nummer `id`. Die Operation blockiert solange bis der Empfänger die Nachricht mit `recv()` entgegennimmt und mit Hilfe von `reply()` eine Antwort schickt. Diese Antwort steht dann in `rbuffer` mit der Maximallänge `rsize` zur Verfügung.

`int recv(void *buffer, size_t size)`
: wartet auf eine Nachricht die in `buffer` mit der Maximallänge `size` abgelegt wird. Der Rückgabewert gibt die Identifikationsnummer des Senders an.

`void reply(int id, void *buffer, size_t size)`
: schickt eine Antwortnachricht an den mit `id` identifizierten Prozess. Diese Funktion soll nicht blockieren und nur erfolgreich sein falls der Zielprozess tatsächlich ein `send()` zu diesem Prozess ausführt bzw. auf dessen Fertigstellung wartet.

`int fork()`
: erzeugt einen neuen Prozess der eine Kopie des aktuellen Prozesses ist. Auch der neue Prozess wird bei Aktivierung aus dem `fork()`-Aufruf zurückkehren. Das einzige Unterscheidungsmerkmal ist die Prozess-Identifikationsnummer. Der Rückgabewert der Funktion ist die Identifikationsnummer des jeweils anderen Prozesses.

Die Puffer für `send()`, `recv()` und `reply()`, sollten am bestem an Seitengrenzen liegen. Eine Möglichkeit, dies statisch zu erreichen, bietet der GCC mit `__attribute__ ((aligned (x)))` und einer Konstanten für `x` an. Anfangs sollten diese Systemaufrufe durch explizites Kopieren der Inhalte der betroffenen Seiten implementiert werden.

3.2 Kopie bei Schreibzugriff (*copy-on-write*)

In einem zweiten Schritt sollen die Seiten nur in den jeweils anderen Adressraum eingeblendet werden und erst im Fall eines Schreibzugriffs physikalisch kopiert werden.
Um Schreibzugriffe zu erkennen und anschließend Seiten kopieren zu können, müssen folgende Mechanismen implementiert werden:

- Einblenden einer Seite in einen anderen Adressraum.
- Betroffene Seiten als nur lesbar (*read only*) in den beteiligten Adressräumen markieren.
- Ein spezieller *pagefault handler* (IRQ 14), der dann die Art des Fehlers feststellt, bei einer nur lesbaren Seite eine physikalische Kopie erzeugt und die Adressraumabbildung entsprechend anpasst.
- Eine Schattentabelle für physikalische Seitenadressen soll die Anzahl der Referenzen auf COW-Seiten zählen. Wenn nur noch eine Referenz übrig ist, muss die Seite bei Schreibzugriffen nicht kopiert werden.

Bei `send`/`recv` bzw. `reply`/`send` sollen die Inhalte der Puffer nur dann mit Hilfe von COW kopiert werden, wenn die Nachrichtengröße mindestens der Seitengröße entspricht (Verschnitt regulär kopieren).

Hinweise:

- Genaue Informationen zum Paging und Pagefaults finden sich im Intel-Handbuch in Kapitel 4.
- `fork` soll natürlich nicht den Kernelbereich mitkopieren.
- Bei COW kann es vorkommen, dass die betroffenen Seiten bereits durch vorherige Übertragungen/`fork()`s COW-abgebildet sind, und die Referenzzähler entsprechend angepasst werden müssen.
- Der TLB wird beim Schreiben von `cr3` implizit geleert. Einzelne Einträge können mit dem Assembler-Befehl `invlpg` invalidiert werden. Dieser Befehl funktioniert nur richtig mit indirekter Adressierung über ein Register: `asm volatile("invlpg (%0)" : : "r"(address) : "memory");`

Abgabe: am 15.07.2015