Zero-Copy

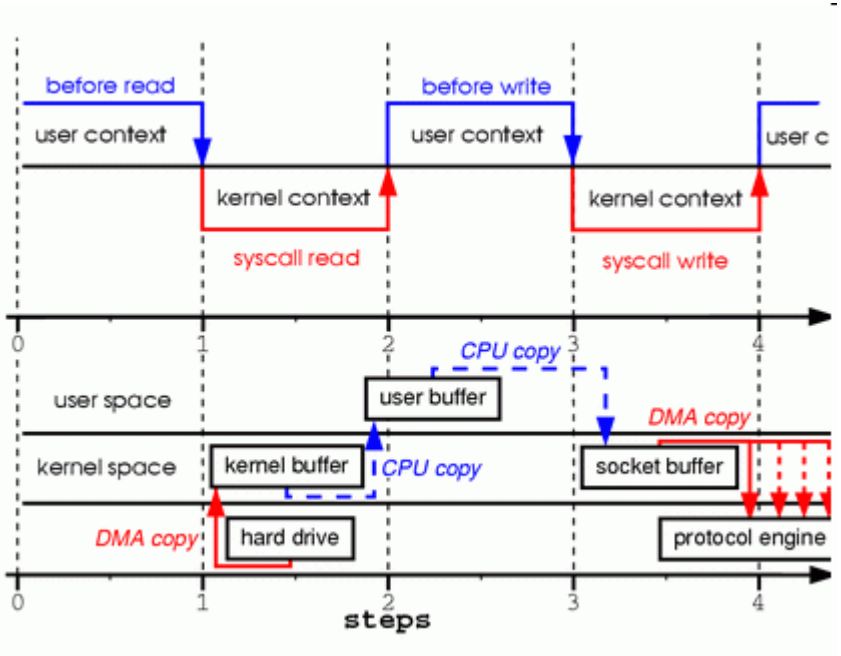
TCP-Optimierung auf Betriebssystemebene

Einfaches Verfahren

- Was passiert im Betriebssystem beim Übertragen einer Datei über einen Socket

```
read(file, tmp_buf, len);
write(socket, tmp_buf, len);
```

Einfaches Verfahren



1. Von der Festplatte in einen Kern-puffer
2. Vom Kern-Puffer in einen User-Puffer
3. Vom User-Puffer in einen Kern-Puffer für Sockets
4. Vom Socket-Puffer in die Protokoll-Implementierung

Insgesamt:

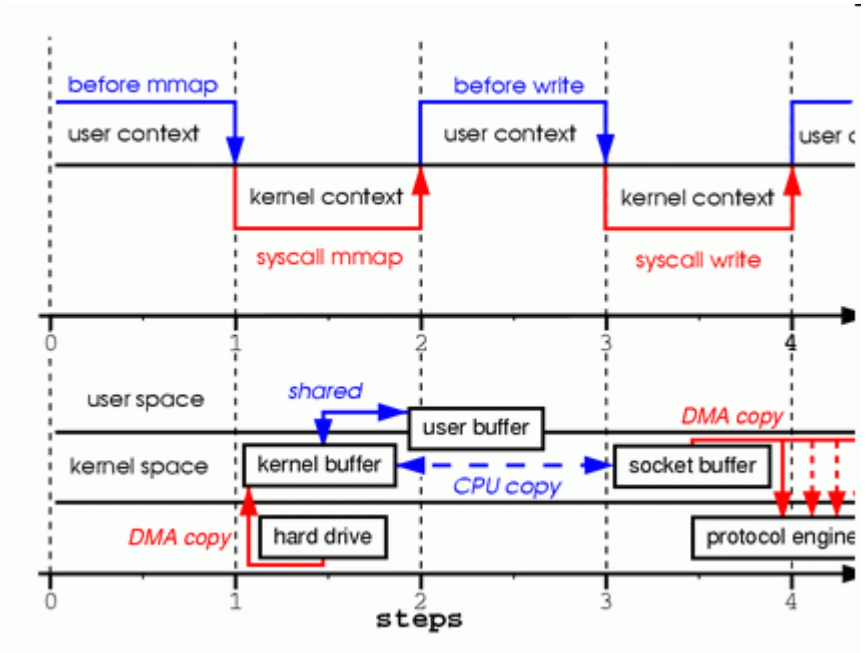
- 4 Kopien
- 3 Kontextwechsel

Memory mapped files

- Verbesserter Ansatz: mmap()

```
tmp_buf = mmap(file, len);
write(socket, tmp_buf, len);
```

Memory mapped files



1. Von der Festplatte in einen Kern-puffer
2. Der Kern-Puffer wird mit dem Userspace geteilt
3. Vom Kern-Puffer in einen Kern-Puffer für Sockets
4. Vom Socket-Puffer in die Protokoll-Implementierung

Insgesamt:

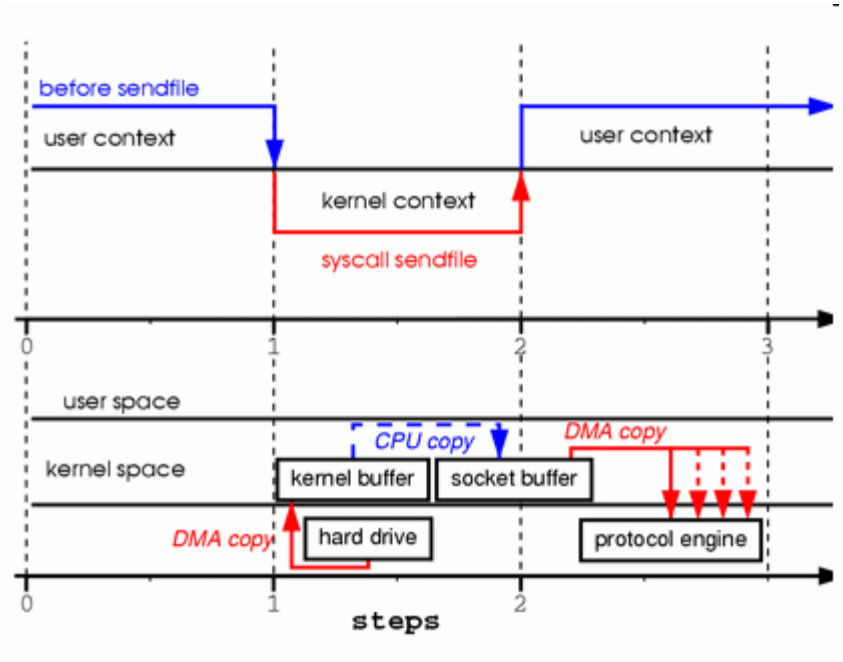
- 3 Kopien
- 3 Kontextwechsel

Sendfile

- Ab Linux 2.1 und in diversen anderen Unices: sendfile()

```
sendfile(socket, file, len);
```

Sendfile



1. Von der Festplatte in einen Kern-puffer
2. Der Kern kopiert selbständig die Daten in einen Socket-Puffer
3. Vom Socket-Puffer in die Protokoll-Implementierung

Insgesamt:

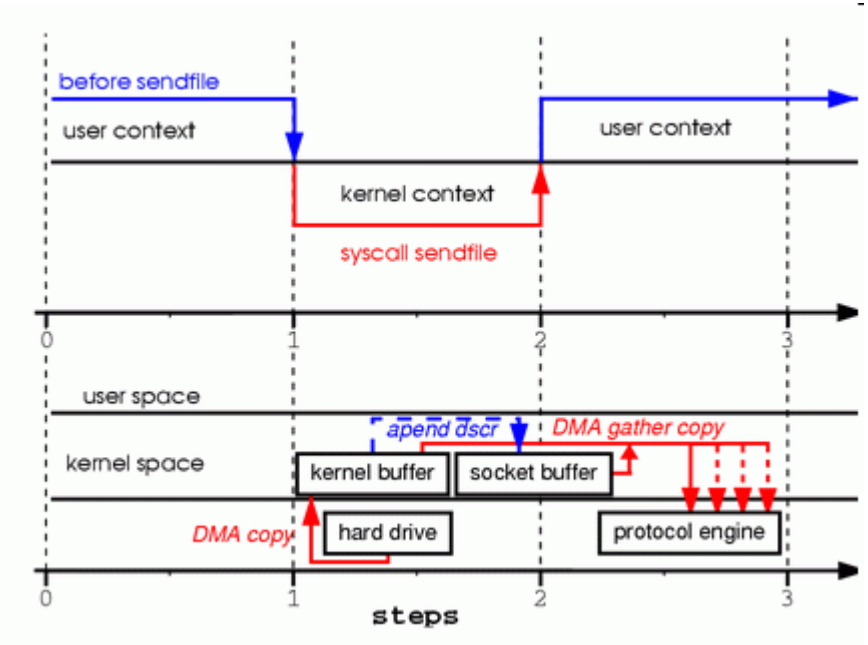
- 3 Kopien
- 1 Kontextwechsel

Sendfile mit Gather

- Ab Linux 2.4 und entsprechende Hardware vorausgesetzt kann noch weiter eingespart werden.
- Aufruf bleibt gleich:

```
sendfile(socket, file, len);
```

Sendfile mit Gather



1. Von der Festplatte in einen Kern-puffer
2. Ein zweiter Puffer mit zusätzlich nötigen Daten wird angelegt
3. Protokoll-Implementierung holt sich Daten aus diesen beiden Puffern selbständig (Gather)

Insgesamt:

- 2 Kopien
- 1 Kontextwechsel