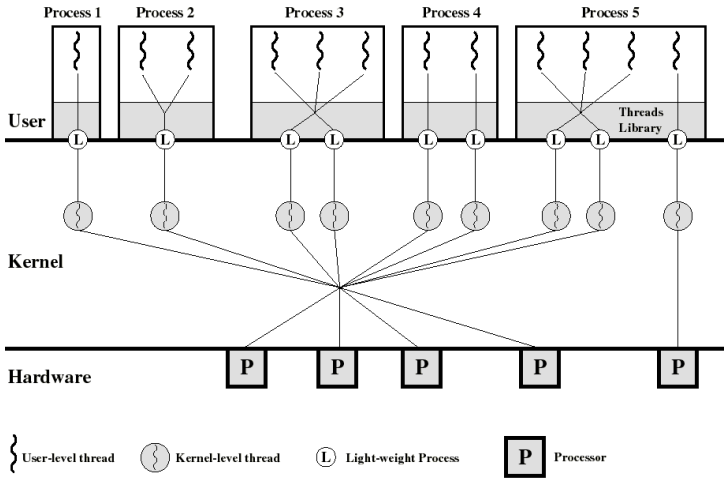
Tim Mallezie 2de bach. III

Architectuur van besturingssystemen: Vraag A5.

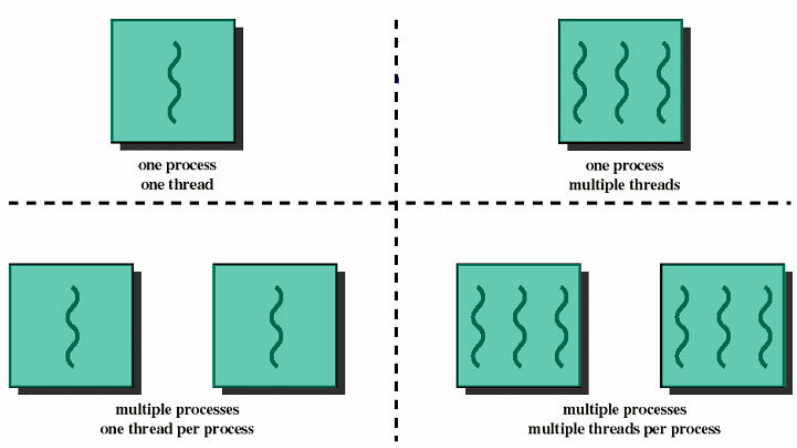
Procesbeheer: *threads*.

a) Wat zijn threads ? Waarom zijn samenwerkende threads binnen hetzelfde proces efficiënter dan samenwerkende processen?
b) Bespreek drie manieren waarop threads kunnen worden geïmplementeerd, en hun voor- en nadelen.
c) Procesbeheer in Windows NT of Solaris: bespreek nevenstaande figuur.
d) Teken en bespreek de threadtoestandsdiagrammen van Solaris.



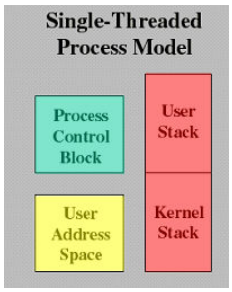
Wat zijn threads?

Een proces kan men definiëren als een eenheid voor de eigendom van bronnen.
Een thread is een eenheid van verdeling van processorinstructies.

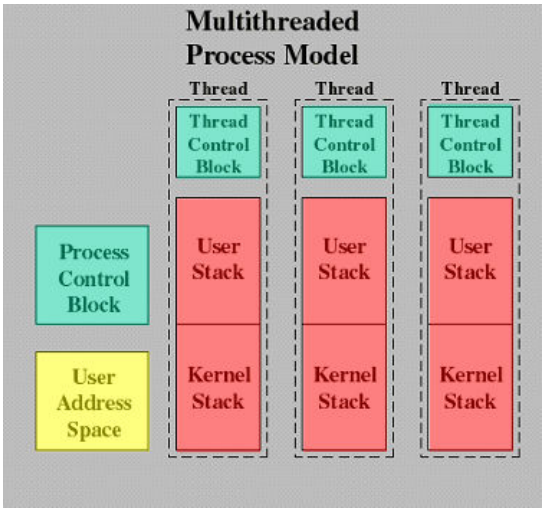


Binnen elk proces kunnen meerdere threads gelijktijdige threads gebruikt worden voor de uitvoering van de instructies. Een multithreaded proces heeft meerdere programmatellers. Alle threads delen alle bronnen die aan het proces zijn toegewezen. Alle threads binnen een proces zijn evenwaardig.

In een procesmodel met één enkel thread bestaat het procesbeeld van een proces uit het PCB, de gebruikersadresruimte voor code en gegevens, en gebruikers- en kernelstacks voor het bijhouden van de procedureaanroepen en de parameters die worden doorgegeven tussen procedures.



Architectuur van besturingssystemen: Vraag A5.



In een omgeving met multithreading is er nog steeds een PCB en gebruikersadresruimte op procesniveau. Er zijn echter voor elke thread ook afzonderlijke stacks en een afzonderlijk besturingsblok, waarin ondermeer de contextinformatie opgeslagen wordt.

Scheduling wordt uigevoerd per individuele thread. De besturingssysteem scheduler maakt geen onderscheid of threads al dan niet tot hetzelfde proces behoren. De meeste toestandsinformatie wordt bijgehouden op threadniveau. Toestanden voor onderbreking zijn echter zinloos op threadniveau: alle threads worden immers tegelijkertijd met het proces uit of terug in het hoofdgeheugen geswapt. Het swappen blijft dan ook beheerd vanuit het PCB.

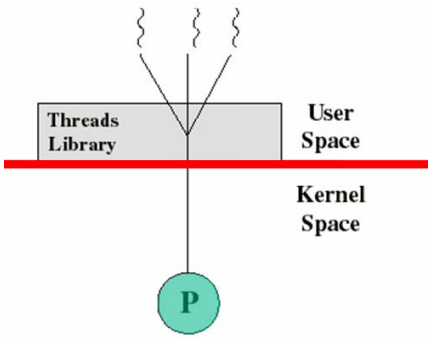
Voordelen van threads.

Het implementeren van een programma op basis van een aantal samenwerkende threads is veel efficiënter dan op basis van processen.

- Threads van hetzelfde proces delen de toestand en bronnen van dat proces: threads hebben gedeelde toestand tot dezelfde geheugengegevens en dezelfde bestanden. Het voordeel van gedeelde code is dat toepassingen meer actieve threads kunnen hebben binnen dezelfde adresruimte waardoor de processor waarschijnlijk beter kan bezig gehouden worden.
- De interprocescommunicatie tussen threads van een zelfde proces kan eenvoudig op gedeeld geheugengebruik worden gebaseerd.
- Het creëren en wisselen van threads binnen een proces vraagt aanzienlijk minder overhead dan de overeenkomstige handelingen op processen.
- Als één thread binnen een proces geblokkeerd wordt hoeft daarom niet het hele proces geblokkeerd worden.

Implementatie mogelijkheden voor threads.

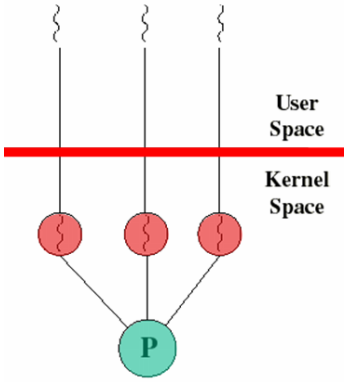
User-level threads:



De kernel is zich niet bewust van het bestaan van user-level threads. Het threadbeheer wordt uitgevoerd door de toepassing zelf. Hierbij wordt gebruik gemaakt van een threadbibliotheek. Dit is een verzameling routines waarmee threads kunnen worden gecreëerd, gewisseld, vernietigd en waarmee de uitvoering van threads kan gescheduled worden. De threadbibliotheek wordt volledig uitgevoerd in de context van het gebruikersproces. De kernel blijft het proces als één geheel beschouwen. De scheduler herkent user-level threads niet.

Voordelen:
User-level threads zijn platformonafhankelijk. Het besturingssysteem moet niets voorzien om threads te ondersteunen.
Threadwisselingen kunnen efficiënter worden doorgevoerd, om dat het proces niet moet overschakelen naar kernelmodus. De overhead van contextwisselingen wordt uitgespaard.
Het scheduling algoritme kan specifiek aan de toepassing worden aangepast.
Nadelen:
Voert één thread een blokkerende systeemaanroep uit, worden alle threads van het proces tegelijkertijd geblokkeerd.
Op elk moment kan maar één enkele thread binnen een proces actief zijn. Een toepassing met user-level multithreading kan geen gebruik maken van multi-processing.

Kernel-level threads:

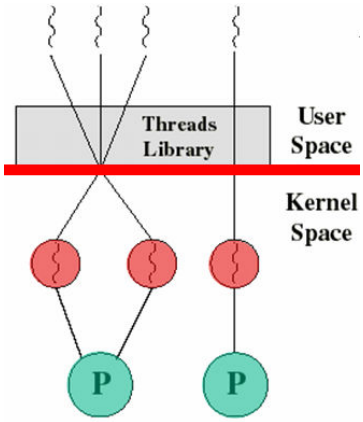


Kernel-level threads worden door het besturingssysteem zelf beheerd. De scheduling door de kernel wordt uitgevoerd op basis van threads, niet op basis van processen. De kernel houdt informatie bij voor het proces in zijn geheel en voor de individuele threads. Een threadwisseling vereist een modus- en contextwisseling. Ontwikkelaars moeten dan ook voorzichtiger omspringen met het aantal threads binnen een proces. Op sommige implementaties wordt om die reden het aantal threads gelimiteerd.
Soms wordt gebruik gemaakt van threadpools. Als een proces start wordt niet één maar meerdere threads aangemaakt, zij worden in een pool geplaatst waar ze wachten op werk. In sommige omgevingen kan het aantal threads in de pool dynamisch aangepast, anders moet gewacht worden tot de pool opnieuw beschikbare threads bevat.

Voordelen:
Indien meerdere processoren beschikbaar zijn, kan de kernel meerdere threads tegelijkertijd uitvoeren.
Een geblokkeerde thread blokkeert het volledige proces niet.
Een threadwisseling vereist een modus- en contextwisseling, maar is toch efficiënter dan een proceswisseling.
Nadelen:
Het besturingssysteem moet voorzien zijn voor de scheduling van kernel-level threads.
Een kernel-level threadwisseling is minder efficiënt dan een user-level wisseling.

Architectuur van besturingssystemen: Vraag A5.

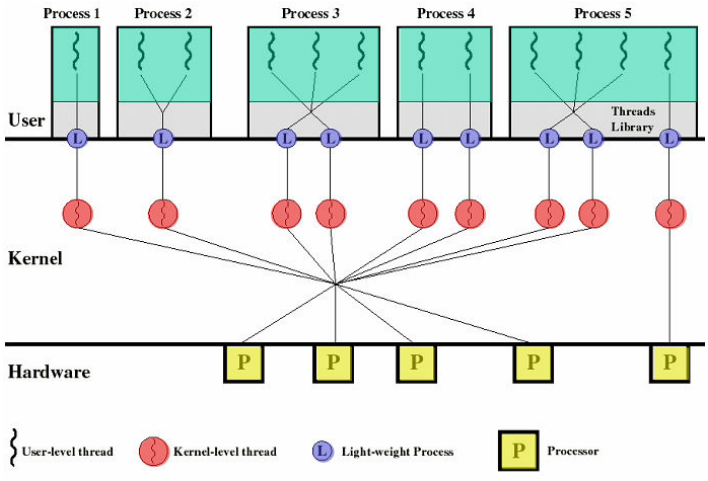
Een combinatie van kernel-level en user-level threads:



Verschillende user-level threads worden gekoppeld aan een kleiner of gelijk aantal kernel-level threads. De user-level threadbibliotheek communiceert met de kernel via lichtgewicht processen of light-weight processen. Voor de bibliotheek ziet een lichtgewicht proces eruit als een virtuele processor, waaraan een user-level thread kan gekoppeld worden. Het besturingssysteem gebruikt het lichtgewicht om er een kernel-level thread aan te koppelen. Als een kernel-level thread blokkeert, blokkeert het lichtgewicht proces ook. Dikwijls kunnen kernel-level threads gebonden worden aan een deelverzameling processen.

Hiermee worden de voordelen van beide implementaties uitgebuit.
Het creëren en wisselen van threads wordt uitgevoerd in de gebruikersruimte, en is dus efficiënter.
Blokkering van een thread kan leiden tot blokkering van een aantal threads binnen het proces, maar niet noodzakelijk tot blokkering van het volledige proces.
Scheduling wordt zowel door de kernel als binnen het toepassingsprogramma uitgevoerd.
Meerdere threads van hetzelfde proces kunnen parallel worden uitgevoerd op multiprocessoren.

Een voorbeeld van gecombineerde user-level en kernel-level threads:

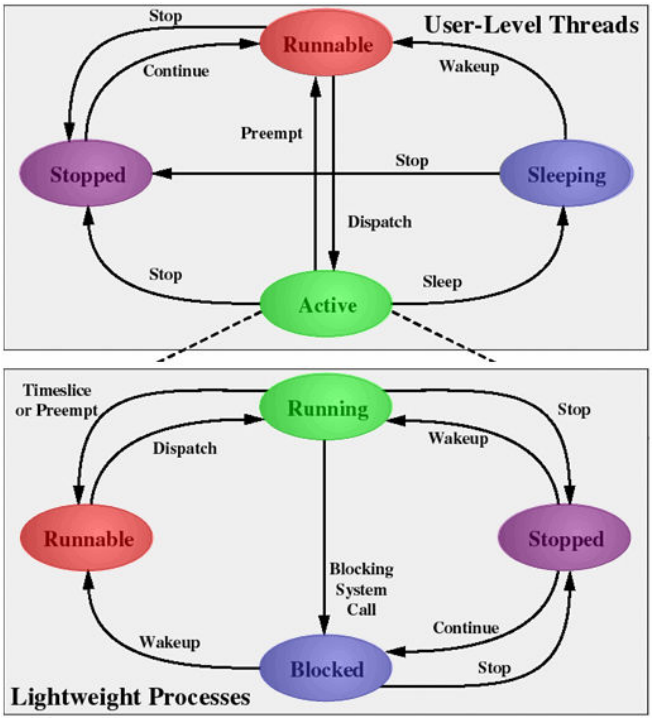


Proces 1: Hier is één user-level thread gebonden aan één lichtgewicht proces. Er bestaat slechts één enkele thread voor uitvoering.
Proces 2: Hier zijn meerdere user-level threads gebonden aan hetzelfde lichtgewicht proces. Dit is een voorbeeld van een zuivere user-level benadering van multithreading.
Proces 3: Hier zijn meerder user-level threads gekoppeld aan een gelijk of kleiner aantal lichtgewicht processen. Dit is een goede oplossing als threads kunnen geblokkeerd worden. De uitvoering van de andere threads binnen hetzelfde proces kan worden voortgezet op de overgebleven lichtgewicht processen.
Proces 4: Hier zijn meerdere user-level threads in een 1-op-1 relatie gebonden aan lichtgewicht processen. Dit is een zuivere vorm kan kernel-level multithreading. De parallelliteit op het kernelniveau wordt volledig zichtbaar voor de toepassing.

Proces 5: Dit is analoog aan proces 3, maar er is aanvullen één user-level thread gebonden aan een lichtgewicht proces, dat gebonden is aan één specifieke processor. Dit wordt gebruikt in toepassing met een realtime component.

Threadtoestanddiagram in Solaris.

De user-level threads vormen op gebruikersniveau de basis interface voor parallelle toepassingen.
De kernel-level threads zijn de entiteiten die kunnen worden gescheduled op een van de processoren van het systeem.
Indien de user-level thread niet gebonden is aan één lichtgewicht proces, dan wordt die enkel in de actieve toestand effectief aan een lichtgewicht proces gekoppeld. Bij het verlaten van de toestand actief, kiest de Green threadbibliotheek een andere user-level thread uit de toestand uitvoerbaar, en wordt die gekoppeld aan het vrijgekomen lichtgewicht proces.
User-level threads verlaten de toestand actief door een primitieve voor synchronisatie aan te roepen waardoor de toestand slapend wordt tot wanneer de voorwaarde voor synchronisatie is voldaan.
Een actieve user-level thread kan ook gestopt worden door een andere actieve user-level thread of door zichzelf.
Een user-level thread kan ook preëmptief onderbroken worden, of zichzelf preëmptief onderbreken.



Ook het lichtgewicht proces kan zich in verschillende toestanden bevinden, de actieve user-level thread wordt pas effectief uitgevoerd wanneer ook het lichtgewichtproces zich in de toestand actief bevindt. Het kan voorkomen dat de user-level thread actief is, maar dat het lichtgewicht proces geblokkeerd is.