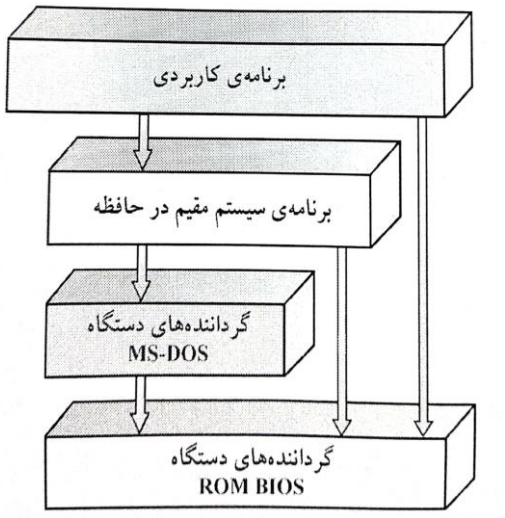
سیستم عامل جلسه ۲

ساختار سیستم عامل

اگر قرار باشد سیستمی به بزرگی و پیچیدگی سیستم عامل به طور مناسب عمل کند و به آسانی تغییر پذیر باشد، باید به دقت مهندسی شود. یک روش متداول، تقسیم وظیفه به مولفه های کوچکتر، به جای یک سیستم بزرگ است. هر یک از این پیمانه ها باید بخش **خوش تعریفی**(Well defined) از سیستم باشند، به طوری که ورودی ها، خروجی ها و عملکردهای آن مشخص باشند. در این بخش، بحث خواهیم کرد که این مولفه ها چگونه با هم ارتباط برقرار می کنند و در هسته با هم ترکیب می شوند.

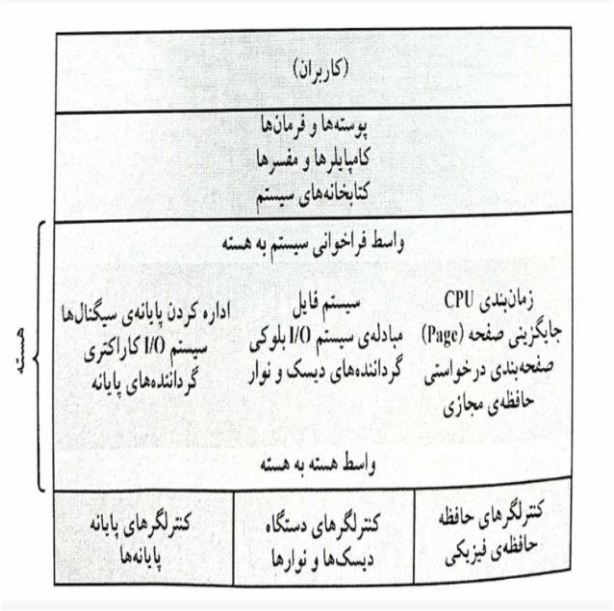
ساختار ساده

اغلب سیستم عامل های تجاری، ساختارهای خوش تعریفی ندارند. غالبا، چنین سیستم هایی به صورت سیستم های کوچک، ساده و محدود شروع به کار می کنند و سپس نسبت به حوزه ی اصلی خود رشد می کنند. MS-DOS مثالی از چنین سیستمی است. این سیستم عامل، ابتدا توسط افرادی طراحی و پیاده سازی شد که ایده ای راجع به شهرت آن نداشتند. این سیستم عامل نوشته شد تا بیشترین قابلیت را در کمترین فضا فراهم سازد و در نتیجه، به دقت به پیمانه ها تقسیم نشده است.



در MS-DOS واسط ها و سطوح عملکرد به خوبی تفکیک نشده اند. برای مثال، برنامه های کاربردی قادرند به روال های I/O پایه نیز دستیابی داشته باشند یا مستقیماً در نمایشگر و گرداننده های دیسک بنویسند. این آزادی، MS-DOS را در مقابل برنامه های مضر آسیب پذیر می سازد و در نتیجه موجب از کار افتادن سیستم می شود. چون ۸۰۸۸ Intel که سیستم عامل MS-DOS برای آن نوشته شد، فاقد **حالت دوگانه** و فاقد حفاظت سخت افزاری است، طراحان MS-DOS هیچ انتخابی نداشتند و سخت افزار را دسترس پذیر رها کردند.

مثال دیگری از ساختار محدود، سیستم عامل یونیکس اولیه است. همانند MS-DOS ، یونیکس ابتدا توسط امکانات سخت افزاری محدود شد. یونیکس شامل دو بخش بود: هسته (کرنل) و برنامه های سیستم. هسته به مجموعه ای از واسط ها و گرداننده های دستگاه تبدیل شد که طی چندین سال اضافه شدند و بسط یافتند. سیستم عامل قدیمی یونیکس را می توان لایه ای در نظر گرفت که در شکل زیر آمده است. هر چیز موجود در پایین واسط فراخوان سیستمی و بالای سخت افزار فیزیکی، هسته است.

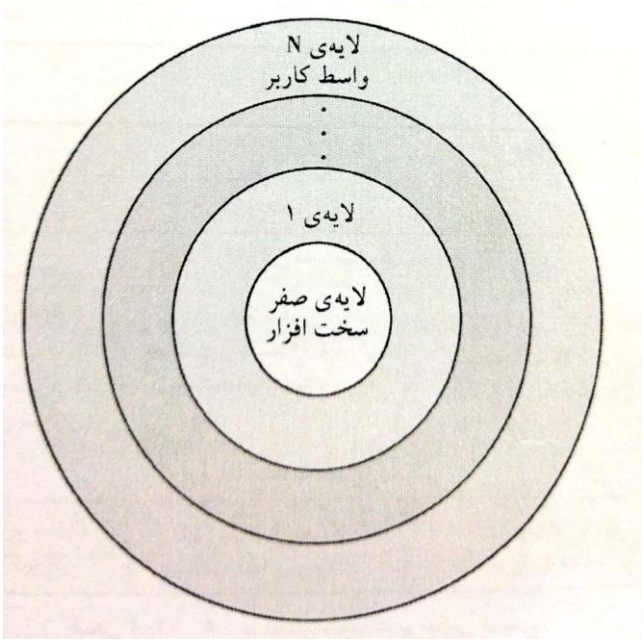


هسته از طریق فراخوان های سیستمی، سیستم فایل، زمان بندی پردازنده، مدیریت حافظه و سایر سیستم عامل را فراهم می سازد. جمع شدن امکانات (عملکردهای) زیاد در یک سطح اشتباه است. پیاده سازی و نگهداری این ساختار یکپارچه دشوار بود هر چند که امتیاز خاصی نیز داشت: در واسط فراخوان سیستم یا در ارتباطات داخل هسته سربار بسیار کمی وجود دارد.

روش لایه ای

با پشتیبانی سخت افزاری مناسب ، این سیستم عامل ها می توانند به مولفه هایی تبدیل شوند که نسبت به سیستم عامل های یونیکس و MS-DOS اولیه، کوچکتر و مناسب تر باشند. در این صورت سیستم عامل روی کامپیوتر و برنامه های کاربردی که از آن استفاده می کنند، کنترل خیلی بیشتری دارد. پیاده سازها ، آزادی بیشتری در تغییر عملکرد داخلی سیستم و ایجاد سیستم عامل های پیمانه ای دارند. تحت روش **بالا به پایین** ، ویژگی ها و عملکرد کلی تعیین و به مولفه هایی تقسیم می شوند. **پنهان سازی اطلاعات** نیز مهم است.

سیستم عامل به چند لایه (سطح) تبدیل می شود. لایه ی پایینی (لایه ی0)، سخت افزار است. لایه ی بالایی (لایه ی N) واسط کاربر است. این ساختار لایه ای در شکل زیر آمده است.



نمونه ای از لایه ی سیستم عامل – مثلاً لایه یM – شامل ساختمان داده ها و مجموعه ای از روال ها است که می تواند توسط لایه های سطح بالاتر فراخوانی شود. لایه ی M ، به نوبه ی خود می تواند عملیات روی لایه های سطح پایین تر را فراخوانی نماید.

امتیاز اصلی روش لایه ای، سهولت ساخت و اشکال زدایی است. لایه ها طوری انتخاب می شوند که هر کدام از توابع (عملکردها) و سرویس های لایه های پایین تر استفاده می کنند. این روش، اشکال زدایی و **وارسی(Verification)** سیستم را آسان می سازد. اولین لایه می تواند بدون توجه به بقیه ی سیستم اشکال زدایی شود، زیرا طبق تعریف، فقط از سخت افزار (که فرض می شود درست است) برای پیاده سازی توابع خود استفاده می کند. وقتی اولین لایه اشکال زدایی

شد، می توان فرض کرد که به درستی عمل می کند و لایه ی دوم اشکال زدایی شود . اگر خطایی در اثنای اشکال زدایی لایه ی خاصی رخ دهد، خطا باید در آن لایه باشد، زیرا لایه های زیر آن قبلاً اشکال زدایی شدند. بنابر این، طراحی و پیاده سازی سیستم، ساده است.

هر لایه فقط توسط عملکردهایی پیاده سازی می شود که توسط لایه های پایین تر فراهم شده اند. لازم نیست یک لایه بداند که این عملکردها چگونه پیاده سازی شده اند.

مشکل عمده ی روش لایه ای، تعریف مناسب لایه های گوناگون است.

مشکل دیگر پیاده سازی های لایه ای این است که نسبت به انواع دیگر کارایی کمتری دارند. برای مثال، وقتی برنامه ی کاربر یک عمل I/O را انجام می دهد، فراخوان سیستمی را اجرا می کند که در لایه ی I/O دچار تله می شود که لایه ی مدیریت حافظه را فراخوانی می کند که آن نیز به نوبه ی خود، لایه ی زمانبند پردازنده را فراخوانی می کند که سپس به سخت افزار ارسال می شود. در هر لایه، پارامتر ها ممکن است تغییر کنند، ممکن است نیاز به ارسال داده ها باشد. هر لایه **سربار**ی را به فراخوان سیستم اضافه می کند. نتیجه اش این است که فراخوان سیستمی در روش لایه ای نسبت به سیستم غیرلایه ای بیشتر طول می کشد.

ریزهسته ها یا ریزکرنل ها

این روش، سیستم عامل را با حذف تمام مولفه های غیر اساسی از هسته و پیاده سازی آن ها به صورت برنامه های سطح کاربر و سیستمی، سازماندهی می کند. نتیجه، هسته ی کوچک تری است. معمولاً، ریز هسته ها، علاوه بر تسهیلات ارتباطی، کمترین مدیریت حافظه و فرآیند را فراهم می کنند.

در اواسط دهه ی ۱۹۸۰، پژوهشگران در دانشگاه Carnegie Mellon، سیستم عاملی به نام Mach را تولید کردند که هسته را با استفاده از روش ریزهسته پیمانه ای کرده است.

پیمانه ها (ماژول ها)

شاید بهترین فناوری برای طراحی سیستم عامل شامل استفاده از پیمانه های بارشدنی هسته باشد. این نوع طراحی، در پیاده سازی های مدرن یونیکس مثل سولاریس، لینوکس و Mac OS X و ویندوز متداول است.

ایده ی این طراحی این است که هسته سرویس های اصلی را ارائه کند در حالی که سرویس های دیگر به طور پویا پیاده سازی می شوند. لینک کردن به سرویس ها به طور پویا، ویژگی های جدیدی را مستقیما به هسته اضافه می کند که لازم است هر وقت هسته تغییر می کند، دوباره کامپایل شود.

نتیجه ی کلی شبیه یک سیستم لایه ای است که هر بخش هسته دارای واسط های تعریف شده و حفاظت شده است؛ ولی از سیستم لایه ای انعطاف پذیرتر است، زیرا هر پیمانه می تواند هر پیمانه ی دیگری را فراخوانی کند. این روش شبیه ریزهسته نیز هست، زیرا پیمانه ی اصلی تنها وظایف اصلی را بر عهده دارد و می داند که پیمانه های دیگر را چگونه بار و با آنها ارتباط برقرار نماید؛ اما کارآمدتر است، زیرا پیمانه ها برای برقراری ارتباط، نیاز به مبادله ی پیام ندارند.

هفت نوع پیمانه ی بارشدنی هسته در سیستم عامل سولاریس :

۱. کلاس های زمان بندی

۲. سیستم های فایل

۳. فراخوان های سیستمی بارشدنی

۴. فرمت های قابل اجرا

۵. پیمانه های STREAMS

۶. پیمانه های متفرقه (گوناگون)

۷. گرداننده های دستگاه و گذرگاه (باس)

سیستم های ترکیبی

اغلب سیستم های عامل، ساختارهای ترکیبی مختلفی دارند و در نتیجه سیستم های ترکیبی به وجود می آیند که مسائل کارایی، امنیت و استفاده پذیری را حل می کنند.

مثال: سیستم عامل Apple Mac OSX و دو سیستم عامل معروف دیگر، یعنی iOS و اندروید.

مفاهیم اساسی زمان بندی

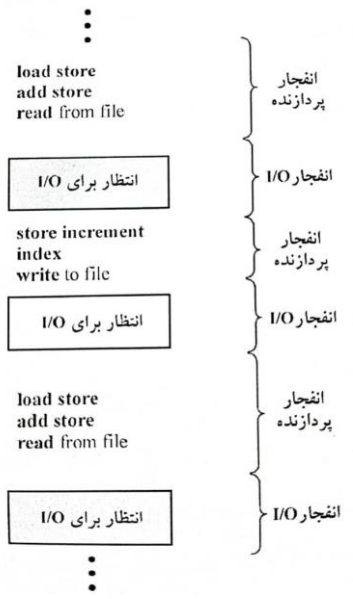
در یک سیستم تک پردازنده ای، در هر زمان فقط یک فرآیند می تواند اجرا شود؛ بقیه باید منتظر بمانند تا پردازنده آزاد شود تا دوباره بتوانند زمان بندی شوند. هدف چند برنامه ای این است که همیشه چندین فرایند در حال اجرا باشند تا بهره وری پردازنده به حداکثر برسید. هر فرایند به اجرایش ادامه می دهد تا برای یک عمل I/O در انتظار بماند. در یک سیستم کامپیوتری ساده، در این مدت پردازنده باید بی کار بماند.

در چند برنامه ای، سعی می شود که از این زمان ها به نحو احسن استفاده شود. در این سیستم، چند فرآیند به طور همزمان در حافظه نگهداری می شوند. وقتی یک فرآیند به حالت انتظار می رود، سیستم عامل پردازنده را از آن فرایند می گیرد و به فرآیند دیگری می دهد. این روند ادامه می یابد ، هر وقت که فرآیندی به حالت انتظار رفت، فرآیند دیگری از پردازنده استفاده می کند.

تقریباً تمام منابع کامپیوتری، قبل از استفاده زمان بندی می شوند. پردازنده یکی از منابع اصلی کامپیوتر است و در نتیجه، زمان بندی آن، موضوع اصلی طراحی سیستم عامل است.

چرخه ی انفجار I/O و انفجار پردازنده

موفقیت زمان بندی پردازنده به این خاصیت فرآیندها بستگی دارد که اجرای فرآیند شامل چرخه ای از اجرای پردازنده و انتظار برای I/O است. فرایندها بین این دو حالت سوییچ می کنند. اجرای فرایند با انفجار پردازنده شروع می شود. به دنبال آن یک انفجار I/O قرار دارد که بعد از آن، انفجار دیگری از پردازنده و سپس انفجار دیگری از I/O قرار دارد و به همین ترتیب ادامه می یابد. سرانجام، آخرین انفجار پردازنده با درخواست سیستم برای خاتمه ی اجرا، پایان می پذیرد.



مدت این انفجارهای پردازنده از فرآیندی به فرآیند دیگر و از کامپیوتری به کامپیوتر دیگر متفاوت است. تعداد زیادی از انفجارهای کوتاه پردازنده و تعداد اندکی از انفجارهای بلند پردازنده وجود دارد. برنامه ی مقید به I/O (در تنگنایI/O) معمولا دارای چند انفجار کوتاه پردازنده است. برنامه ی مقید به پردازنده (در تنگنای پردازنده) معمولا چند انفجار بلند پردازنده دارد. این توزیع می تواند در انتخاب الگوریتم زمان بندی پردازنده مهم باشد.

زمانبندی پردازنده

هر وقت پردازنده بی کار می شود، سیستم عامل باید یکی از فرآیندهای موجود در صف آماده را برای اجرا انتخاب کند. این انتخاب توسط زمان بند کوتاه مدت (یا زمان بند پردازنده) انجام می گیرد. زمان بند، فرایندی را از بین فرآیندهای موجود در حافظه که آمادگی اجرا دارند انتخاب می کند و پردازنده را به آن تخصیص می دهد.

که صف آماده، الزاما یک صف با ویژگی "خروج به ترتیب ورود" (FIFO) نیست . همان طور که هنگام بحث در مورد الگوریتم های زمان بندی خواهیم دید، صف آماده می تواند به صورت صف FIFO، صف اولویت، درخت یا یک لیست پیوندی نامرتب پیاده سازی شود. از نظر مفهومی تمام فرآیند های موجود در صف آماده، منتظر به دست آوردن پردازنده هستند تا اجرا شوند.

زمان بندی با قبضه کردن(Preemptive)

تصمیمات زمان بندی پردازنده ممکن است تحت چهار شرط زیر اتخاذ شود:

۱. وقتی که فرآیندی از حالت اجرا به حالت انتظار می رود (مثل درخواست I/O، یا فراخوانی wait() برای خاتمه ی یکی از فرآیندهای فرزند).

۲. وقتی فرآیندی از حالت اجرا به حالت آماده می رود (مثل وقتی که رویدادی رخ می دهد).

۳. وقتی که فرآیندی از حالت انتظار به حالت آماده می رود (مثل تکمیل I/O)

۴. وقتی که فرآیندی خاتمه می یابد.

برای شرایط ۱ و ۴، انتخابی بر حسب زمان بندی وجود ندارد. در صورت وجود فرایندی در صف آماده ، یک فرآیند باید برای اجرا انتخاب شود. اما برای شرایط ۲ و ۳ امکان انتخاب وجود دارد.

وقتی زمان بندی فقط تحت شرایط ۱ و ۴ انجام می گیرد، الگوی زمان بندی را **بدون قبضه کردن(Nonpreemptive)** می نامیم و در غیر این صورت، آن را **با قبضه کردن(Preemptive)** می نامیم. در زمان بندی بدون قبضه کردن، وقتی پردازنده به فرآیندی تخصیص یافت، آن قدر پردازنده را نگه می دارد تا این که خاتمه یابد یا به حالت انتظار برود که در این صورت پردازنده را آزاد می کند. این روش زمان بندی توسط ویندوزx3. مورد استفاده قرار می گرفت؛ ویندوز ۹۵، زمان بندی با قبضه کردن را معرفی کرد و تمام نسخه های بعدی سیستم عامل ویندوز از زمان بندی با قبضه کردن استفاده نمودند. سیستم عامل Mac OSX برای مکینتاش نیز از زمان بندی با قبضه کردن استفاده می کند؛ نسخه های قبلی سیستم عامل مکینتاش مبتنی بر **زمان بندی همکاری**(Cooperative scheduling) بود. زمان بندی همکاری تنها روشی است که می تواند بر روی سکوهای سخت افزاری اجرا شود، زیرا به سخت افزار خاصی (مثلا تایمر) که برای زمان بندی با قبضه کردن نیاز است، نیاز ندارد.

زمان بندی قبضه کردن هزینه دارد. حالتی را در نظر بگیرید که در فرآیند داده های مشترکی دارند. یکی از آن ها ممکن است در زمان به هنگام سازی داده ها قبضه شود و پردازنده ی دوم در حال اجرا باشد. فرآیند دوم ممکن است تلاش کند داده هایی را بخواند که در حالت ناسازگار قرار دارد. بنابراین، نیاز به راهکار جدیدی است تا دستیابی به داده های مشترک را هماهنگ کند.

زمان بندی با قبضه کردن در طراحی هسته ی سیستم عامل موثر است. هنگام پردازش فراخوان سیستم، هسته ممکن است مشغول انجام فعالیتی در رابطه با یک فرآیند باشد. این فعالیت ها ممکن است داده های مهم هسته را تغییر دهند (مثل صف I/O).

اگر فرآیندی که در حال انجام این تغییرات است، قبضه شود و هسته (یا گرداننده ی دستگاه نیاز به خواندن یا تغییر همان ساختار داشته باشد، چه اتفاقی می افتد؟

به نظر می رسد که هرج و مرج پیش می آید. بعضی از سیستم های عامل، از جمله اغلب نسخه های یونیکس ، این مساله را به این ترتیب حل می کنند که قبل از تعویض متن، منتظر می مانند تا فراخوان سیستم کامل شود یا یک عملI/O رخ دهد. این الگو تضمین می کند که ساختار هسته ساده باشد، زیرا درحالی که ساختمان داده های هسته در وضعیت ناسازگاری وجود دارند، هسته فرآیندی را قبضه نمی کند. متاسفانه ، این مدل اجرای هسته برای پشتیبانی محاسبات بی درنگ و چند پردازشی مناسب نیست.

چون وقفه ها در هر زمانی می توانند رخ دهند و چون هسته همیشه نمی تواند آن ها را نادیده بگیرد، بخش هایی از کد که تحت تأثیر وقفه ها قرار دارند، باید از استفاده ی همزمان در امان باشند. لازم است سیستم عامل وقفه ها را در تمام اوقات بپذیرد، و گرنه ممکن است ورودی مفقود شود و خروجی از بین برود. برای این که چندین فرایند به طور همزمان به این

بخش های کد دسترسی نداشته باشند، هنگام ورود به آن ها وقفه را غیر فعال و هنگام خروج، وقفه را فعال می سازند. توجه به این نکته مهم است که بخش هایی از کد که وقفه ها را غیر فعال می کند زیاد نیستند و دستورالعمل های زیادی ندارند.

توزیع کننده

مولفه ی دیگری که در زمان بندی پردازنده دخالت دارد، **توزیع کننده(Dispatcher)** است. توزیع کننده ، پیمانه ای است که کنترل را به پردازنده ای می دهد که زمان بند کوتاه مدت آن را انتخاب کرده است.

این عمل شامل موارد زیر است:

- **تعویض متن (تعویض بستر)**
- **تغییر به حالت کاربر**
- **پرش به محل مناسبی در برنامه ی کاربر و آغاز مجدد آن برنامه**

توزیع کننده باید سرعت بسیار بالایی داشته باشد، زیرا هنگام تعویض هر فرایند فراخوانی می شود. مدت زمانی که توزیع کننده صرف می کند تا یک فرآیند را متوقف و فرآیند دیگری را شروع کند **تأخیر توزیع(Dispatch latency)** نام دارد.

معیارهای زمان بندی

- **بهره وری پردازنده:** می خواهیم تا آنجایی که ممکن است پردازنده مشغول باشد. درصد بهره وری پردازنده می تواند از صفر تا ۱۰۰ باشد. در یک سیستم واقعی، باید از ۴۰ (برای سیستم هایی با بار کم) تا ۹۰ درصد (برای سیستم هایی با بار زیاد) باشد.
- **توان عملیاتی(Throughput) (گذردهی):** اگر پردازنده مشغول اجرای فرآیندها باشد، کاری در حال انجام است.یک معیار کار، تعداد فرآیندهایی است که در واحد زمان کامل می شوند و توان عملیاتی نام دارد. برای فرایند های طولانی، این نرخ ممکن است بر حسب ساعت سنجیده شود. برای تراکنش های کوتاه توان عملیاتی ممکن است ۱۰ فرآیند در ثانیه باشد.

- **زمان برگشت(Turnaround time) (یا زمان کل):** از نقطه نظر یک فرآیند خاص، مهم ترین معیار مدت زمان اجرای یک فرایند است. فاصله ی زمانی از زمان تحویل فرآیند، تا زمان کامل شدن اجرای آن ، زمان برگشت نام دارد. زمان برگشت مجموع زمان انتظار برای قرار گرفتن در حافظه، زمان انتظار برای قرار گرفتن در صف آماده، اجرا در پردازنده و عمل I/O است.
- **زمان انتظار:** الگوریتم زمان بندی پردازنده، بر مدت زمانی که در اثنای آن فرآیندی که اجرا می شود یا عمل I/O انجام می دهد، تأثیر ندارد؛ فقط بر مدت زمانی که فرآیند در صف آماده منتظر می ماند موثر است. زمان انتظار برابر با مجموع مدت های انتظار یک فرآیند در صف آماده است.
- **زمان پاسخ:** در یک سیستم محاوره ای، زمان برگشت ممکن است معیار خوبی نباشد. اغلب، یک فرآیند ممکن است بعضی از نتایج را زود تولید کند و هنگامی که این نتایج به خروجی می روند، محاسبات دیگری صورت گیرد و نتایج دیگری به خروجی برود. لذا، معیار دیگر، زمان تحویل یک درخواست تا دریافت اولین پاسخ است. این معیار که زمان پاسخ نام دارد، مدت زمانی است که طول می کشد پاسخ آماده شود، نه مدت زمانی که طول می کشد تا آن پاسخ چاپ شود. زمان برگشت، به سرعت دستگاه خروجی بستگی دارد.